Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedésmérnöki és Járműmérnöki Kar
Közlekedés- és Járműirányítási Tanszék

Intelligens jelzőfej alkalmazása a közúti forgalomirányításban

Jenes Géza
KK5QP1

Egyetemi konzulens:
Dr. Tettamanti Tamás
Közlekedés- és Járműirányítási Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Külső konzulens:
Ludvig Ádám

2013. december 11.

Absztrakt:

A dolgozat egyik fő célja az intelligens jelzőfejekkel működő rendszer felépítése volt a biztonság kérdésének szem előtt tartásával. Különböző rendszerkomponensek működésbe hozásával, és programozással épült fel az intelligens jelzőfejjel működő forgalomirányítási rendszer. Gazdasági számítások igazolták a projekt pénzbeli megtérülését, és kiépítésbeli előnyét a hagyományos forgalomirányítással szemben.

Kulcsszavak:

közúti forgalomirányítás, jelzőlámpa, intelligens jelzőfej, PLC, Actros VTC 3000

Köszönetnyilvánítás

Köszönöm konzulenseimnek Tettamanti Tamásnak és Ludvig Ádámnak a dolgozatom elkészítésében nyújtott rendkívül sok segítséget, tanácsaikat és hasznos észrevételeiket. A Budapesti Műszaki és Gazdaságtudományi Egyetem Közlekedés- és Járműirányítás Tanszékének az általuk biztosított eszközöket. A Swarco Traffic Hungaria Kft.-nek a rendelkezésemre bocsátott csomópont kialakítási terveket. Szabó Gézának, Aradi Szilárdnak és Mátyus Gergelynek az elektrotechnikai tanácsokat és segítséget.

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

1. fejezet

Bevezető

A közlekedés bármely területét vizsgálva elmondhatjuk, hogy a költségek csökkentése mindig a kitűzött célok között szerepel. Természetesen ez nem mehet a minőség és a hatékonyság rovására, hanem egy olyan új állapot létrehozását kell jelentse, amely a régit teljes mértékben helyettesíti, és akár tudásában is túlmutat rajta. A tudomány előrehaladásával néha a meglévő, bár megbízható rendszereket, érdemes újakkal helyettesíteni, amelyek a jelenkor technikája által nyújtott előnyöket jobban kihasználják.
A dolgozatom célja, hogy egy napjainkban jól működő rendszert, egy új, korszerű komponens bevonásával egy kicsit átalakítsa, és vele járó előnyöket, hátrányokat a régi rendszerhez viszonyítva megvizsgálja. Témája a jelzőfej „intelligensé" tételét tanulmányozza. A jelzőfej, mely csak a jelzésre szolgáló izzókat hordozza magán, úgy kaphat valamilyen intelligenciát, ha az izzók kapcsolási vezérlését átveszi, továbbá a feszültség- és áramellenőrzés funkciókat logikailag továbbítja a vezérlő berendezés irányába. Egy jelzőfejhez kapcsolt számítógép képes ilyen feladatok elvégzésére. Így a jelzőfej egy olyan eszközzé válik, melynek konkrét feladata is van a rendszerben és nem csak kimenetként szolgál. Az általam készített projekt a forgalomirányító berendezés és a jelzőfejek kábeles kapcsolatában hoz áttörést, ugyanis képes ezt kiváltani egy egyszerűbb, olcsóbb kapcsolatra. Ez természetesen sok munkát von maga után, ugyanis a jelzőfejre egy kis logikai modult kell építeni, és a jelzőfej vezérlését teljesen újra kell tervezni. Egy átalakítás mindig sok problémával jár, melyekkel én is szembesültem a fejlesztés során.
A szakdolgozatom során nem csak elméleti kutatást végeztem, hanem az intelligens jelzőfej fizikai megvalósítását is elvégeztem (Actros típusú forgalomirányító berendezés, PLC-k, és LED-es jelzőfejek segítségével). Dolgozatomban ezen új rendszer, azaz az intelligens jelzőfejekkel történő forgalomirányítás feltételeiről, megvalósításáról, az új rendszerben fellépő problémákról és azok kezeléséről írtam. Továbbá foglalkoztam még az új rendszer biztonságával és hatékonyságával, valamint összehasonlítottam a hagyományos forgalomirányítással.

2. fejezet

A közúti jelzőlámpás forgalomirányítás

A közúti közlekedés irányítása a felgyorsult világban döntő szerepet tölt be. A közlekedés egyik fő feladata, hogy az emberek helyváltoztatási igényeit kielégítse, így egyik legfontosabb hatása, hogy befolyásolja az emberi időfelhasználást. Szükségessége lehetővé teszi a lakóhely és a munkahely elkülönülését, kielégíti a termelés, az elosztás és a fogyasztás szállítási, utazási igényeit, továbbá elősegíti a szabadidő felhasználását. [Debreczeni, 2013]
A helyváltoztatási igény egyre nő, így az utakon megjelenő járművek száma is növekszik. A közforgalmú közlekedést nem elegendően veszik igénybe, és nagyon sokan egyedül utaznak egy járműben, ezért a közúti járműforgalom nagysága jelentősen megugrott az utóbbi évtizedekben. Ez magával ránt több problémát is, mint például fokozott balesetveszélyt, megnövekedett környezet terhelést, forgalmi torlódásokat. Az eljutási idő és az átlagsebesség csökken, a megállások száma nő. Az emberekben emiatt sokszor nő a stressz, ami balesetekhez vezethet. Ezeket a problémákat valahogy kezelni kell, hogy a városi közlekedés gördülékenyebben működjön. Ahhoz, hogy elkerüljük az utakon történő baleseteket, csökkentsük a torlódásokat, illetve azok externáliás hatásait, továbbá az emissziót és a zajt is visszafogjuk, a közúti közlekedést irányítani kell. A jelzőlámpás szabályozás lokális vagy hálózati szintű alkalmazása hozzájárulhat egy zavartalanabb forgalomlebonyolódáshoz. A különböző forgalmi irányok közötti kapcsolat a csomópontokban jelenik meg, ezért azt mondjuk, hogy ez a leggyengébb láncszem. Itt a szűk keresztmetszet miatt megjelenik a forgalmi akadályoztatás, ami időveszteséget, fokozott balesetveszélyt és jelentős környezetterhelést von maga után.[Luspay et al., 2011]
A jelzőtáblák, jelzőlámpák és az útburkolati jelek azok a szabályozási eszközök, melyeket a közúti közlekedésben kialakított szabályok közül adott időben, helyen, helyzetben, illetve járművel és gyalogosként is be kell tartani [Debreczeni, 2013]. A közlekedésben résztvevőkre vonatkozó szabályokat, előírásokat, törvé-

nyeket és rendeleteket hazánkban a KRESZ foglalja össze. Jelzőlámpás irányítást csak olyan esetben alkalmazhatunk, ha adott csomópont a jelzésképek hiányában jelzőtáblás irányításra is alkalmas. A jelzőlámpás irányítás alkalmazásának indokoltsága összetett kérdés, de van néhány irányadó alapelv. Első és legfontosabb szempont, ha egy adott csomópontban rendszeresen hasonló típusú balesetek történnek. Ilyenkor javasolt a jelzőlámpa bevezetése. Továbbá főútvonalak csomópontjain, négy és több forgalmi sávú utak találkozásakor, ha hosszú a várakozási idő az alárendelt úton, vagy mindkét irányban villamos közlekedik, illetve veszélyes gyalogátkelőhely esetén. Összehangolt rendszer kialakításánál egy adott hálózaton is elengedhetetlen a jelzőlámpás irányítás. A jelzőlámpákat több szempont alapján lehet csoportosítani:

- szabványos lencseátmérő alapján: 100 mm, 200 mm, 300 mm – ezt a láthatóság határozza meg,
- tele zöld vagy maszkos zöld (lásd 2.1 ábra),

2.1. ábra. Tele zöld és maszkolt zöld lámpák [Debreczeni, 2013]

- három-, kettő- vagy egyfogalmú készülék (lásd 2.2, 2.3, és 2.4 ábrák),

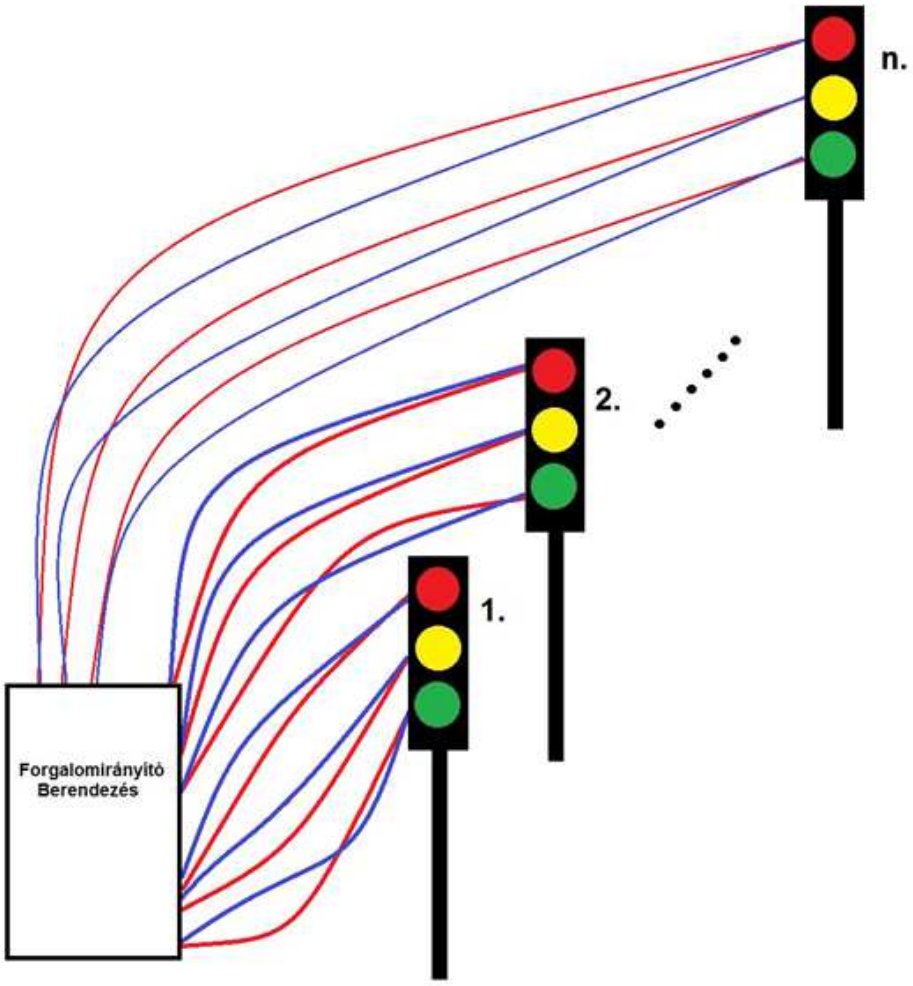
2.2. ábra. Háromfogalmú jelzőberendezések [Debreczeni, 2013]

2.3. ábra. Kétfogalmú jelzőberendezések [Debreczeni, 2013]

2.4. ábra. Egyfogalmú jelzőberendezések [Debreczeni, 2013]

- a jelzőlámpák világítóteste alapján: hagyományos izzó vagy korszerű világító dióda (LED: Light Emitting Diode [Sakai and Kawamura, 1987]).

A jelzőlámpás irányítástechnika eszközei a jelzőfejek, valamilyen forgalomirányító berendezés, valamint a köztük lévő kommunikációhoz szükséges átviteli közeg. A dinamikusan vezérelt csomópontoknál szükség van valamilyen járműérzékelő detektorra is. A napjainkban működő rendszereknél a jelzőfejek közvetlen, kábeles kapcsolatban állnak a forgalomirányító berendezéssel. Minden egyes jelzőfej, minden izzójához ki kell húzni egy vezetéket és visszavezetni forgalomirányító berendezésbe, hogy az áramkörön, mint ellenállás működjön az izzó. Ez a fajta kiépítés rengeteg vezeték szükségességét vonja maga után a jelzőlámpával irányított kereszteződésben (lásd 6.7 ábra). Napjainkban működő jelzőfejek vegyesen hagyományos izzóval illetve LED izzóval felszerelt eszközök. A LED-del szerelt jelzőfejeknek számos előnye van. Nincs betekintési szögből eredő fénytévesztés, hosszabb az élettartam, gazdaságosabb az üzemeltetés. Az izzókkal szemben ezeket a rendszereket általában nem 230V-ról, hanem 40V-ról kell táplálni, ami többször problémát jelentett például a budapesti LEDesítési projektben. Léteznek 230V-os LED-es izzók is, de a LED-es jelzőfejek fő előnye az érintésvédelmi szempontból, hogy kis feszültségről üzemelnek, így ma már leginkább a 40V-os megoldást választják. Egy 40V-ról működő rendszer viszont sokkal érzékenyebb például a felszültség ellenőrzésre, ami a jelzőlámpás ellenőrzés egyik alap művelete. Ezzel kell a zöld égését, árammal pedig a piros izzó megfelelő működését ellenőrizni. Alacsonyabb



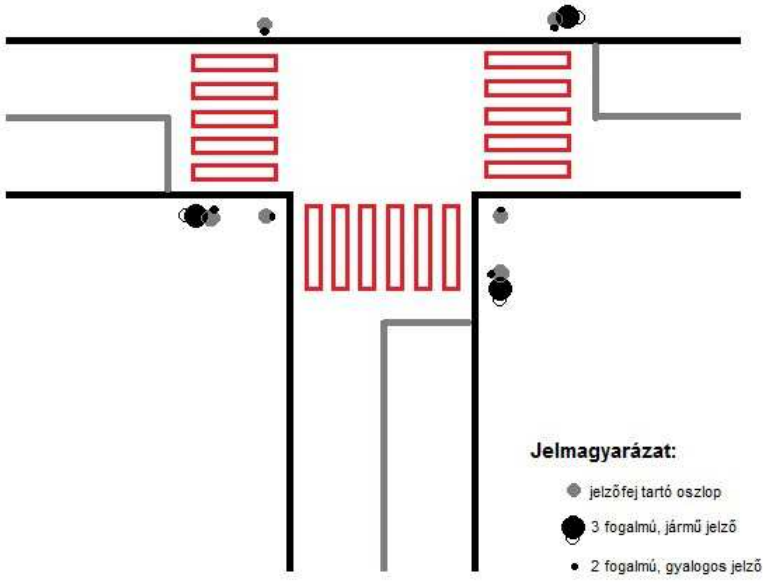
2.5. ábra. A jelzőlámpás forgalomirányítás kábelezése sematikusan

feszültségen 1-2V-os kilengést már nem lehet elhanyagolni, míg például a 230V-os meghajtásnál ez nem jelent problémát. Ez azt eredményezi, hogy sokkal jobb minőségű kábelezést igényel ez a rendszer. Például egy felújítást végző projektnél nagy problémát tud eredményezni, hogyha a megrendelő csak a jelzők cseréjét tervezte, de kiderül, hogy a föld alatt lévő teljes kábelezést cserélni kell. Ezen rendszerek további hátránya, hogy burkolatjavítási munka során gyakran elvágják az alépítményi vezetékeket, amelyekből egy nagyobb csomópont esetén igen sok fut a földben. Ez súlyos károkat tud eredményezni, hiszen minden átvágott kábelt javítani vagy pótolni kell.

3. fejezet

Az intelligens jelzőfej alkalmazásának lehetősége a közúti forgalomirányításban

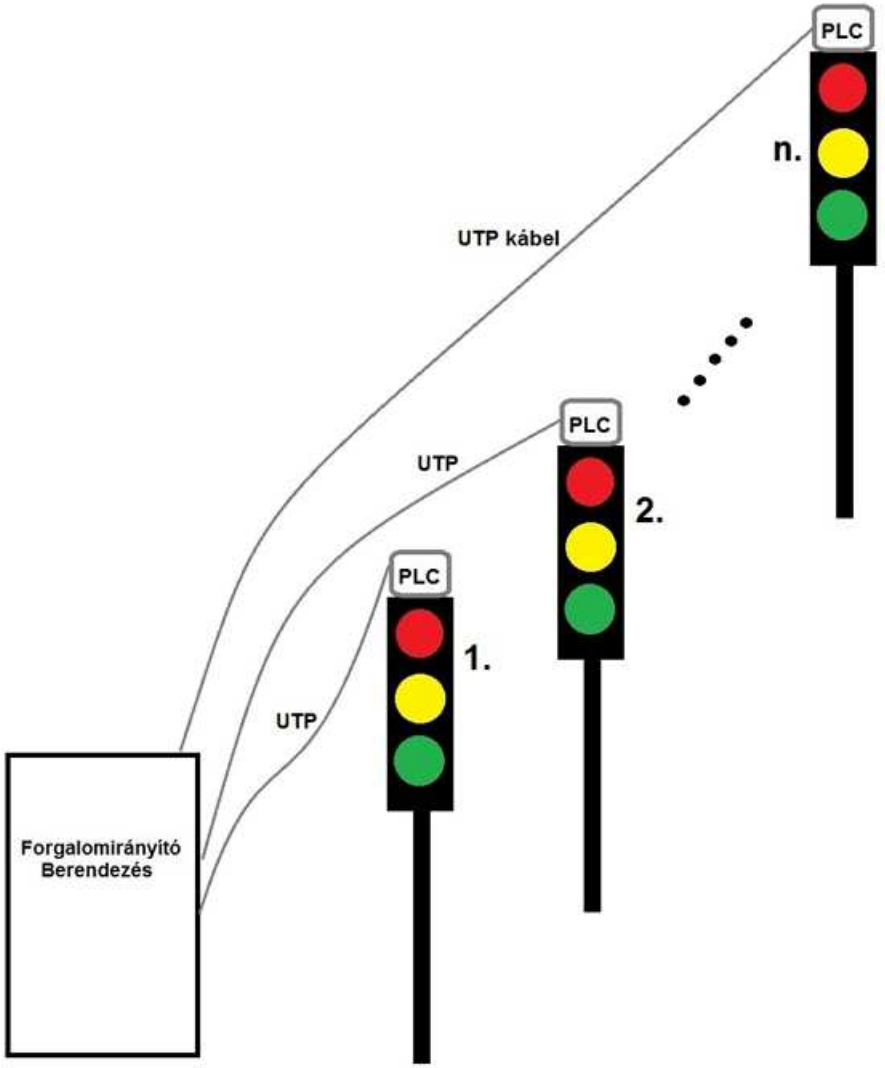
Dolgozatom alap ötlete, hogy a jelzőfejek és a forgalomirányító berendezés közötti kábeles kapcsolatot valamilyen módon egyszerűsítsem, a vezetékek számát és hosszát redukáljam. A jelzőlámpás forgalomirányítás általánosságban egy irányító berendezésből és jelzőfejekből tevődik össze. Ebben a struktúrában minden működéshez kapcsolódó feladatot (irányítás, kommunikáció, ellenőrzések) a forgalomirányító berendezés lát el. A jelzőfejek feladatköre kizárólag a jelzésképek megjelenítése. Ez egy logikus kialakítás, hiszen a rendszer biztonsága könnyebben és jobban ellenőrizhető, ha csak egy komponens végez irányítási feladatokat. Ugyanakkor az egyes elemek közötti kapcsolat meglétéhez rengeteg kábelezésre van szükség, amely egy nagyobb csomópont esetén elég nagy költségeket tud maga után vonni. Egy vezeték mérete 4-től akár 30 erűig is terjedhet, mely az árukat nagyban meghatározza. Az érszámot azon ponton lévő jelzőfejek száma határozza meg, ahova a kábelt ki kell húzni.
Vizsgáljunk meg a példa kedvéért egy egyszerű T alakú csomópontot gyalogátkelőhelyekkel! A 3.1 ábrán látható kereszteződésben minimum 3 közúti háromfogalmú jelzőberendezés megléte szükséges a forgalomirányításhoz (és ez a legegyszerűbb eset, mert nincs ismétlőjelző belógatva az út fölé), továbbá 6 kétfogalmú jelzőberendezés a gyalogos átkelőhelyek irányításához. Ehhez a 9 jelzőhöz mind több erű vezetéket kell kihúzni. Ha csak darabszámra számoljuk, 9 x 3, azaz 27 vezetékre van szükség, és akkor még nem is néztük a vezetékek méretezését és hosszát, amit a forgalomirányító berendezéshez képesti helyük határoz meg.
A projektemben bevezetem az intelligens jelzőfejeket a rendszerbe. Célom megvalósításához Siemens PLC (Programmable Logic Controller [Siemens, 2009]) eszközöket használok fel. Ez a programozható logikai vezérlő átveheti a jelzőfejek izzói-



3.1. ábra. Egy általános forgalomirányítású, T alakú csomópont sematikusan

nak kapcsolási feladatát a forgalomirányító be-rendezéstől, hiszen programozható relékimenetekkel rendelkezik. Az általam tervezett rendszerben, minden jelzőfejhez egy ilyen PLC-t csatlakoztattam (azonos jelzésképet adó jelzőfejeknél elég egy darab vezérlő egység is), melyek kapcsolatban állnak a csomópont forgalomirányító berendezésével, de az izzók kapcsolgatását ők maguk végzik. Ezzel a koncepcióval megszüntethetem a fölösleges vezetékezést, hiszen a jelzőfejek izzóit a PLC-kbe kötöm be, mely magán a jelzőfejen kap helyet, így az elektromos vezetékek hossza ott csupán 20 – 30 centiméterre redukálódik. A forgalomirányító berendezés és a PLC kapcsolatához pedig mindösszesen egy darab UTP (Unshielded Twisted Pair) kábelt kell csak kihúzni (lásd 3.2 ábra). Természetesen a logikai modulok tápellátását is biztosítani kell, de erre elég egy gerinchálózaton végigvezetett 4 erű kábel.
Jelen rendszerben a TCP/IP (Transmission Control Protocol/Internet Protocol) protokollt alkalmaztam a hálózati kommunikáció felállítására, de ez történhetne bármely egyéb protokollal is. Szakdolgozatomban egy megoldást kínálok az intelligens jelzőfejjel működő rendszer kiépítésére. Ez jövőben átalakítható, illetve továbbfejleszthető más megoldás kipróbálásának céljából.
Ezzel jelentős költségmegtakarítást érhető el, mert a több erű kábelek, csak a logikai modul és az izzó közé szükségesek, ami pár tíz centiméteres szakasz. A forgalomirányító berendezés és a logikai modul közé pedig egy jóval olcsóbb UTP kábel kerül. A megoldás előnyös lehet az ideiglenesen alkalmazott jelzőlámpás

forgalomirányításban is - például forgalmi akadály vagy útépítés miatt kihelyezett jelzőknél. Ilyenkor az ideiglenes rendszer telepítése sokkal gyorsabb és egyszerűbb lehet. Forgalmi terelés esetén akár a vezérlő gép is elhagyható, mert annyira egyszerű a feladat, hogy azt képes a PLC egymagában megoldani.



3.2. ábra. Az intelligens jelzőfejjel történő forgalomirányítás koncepciója

4. fejezet

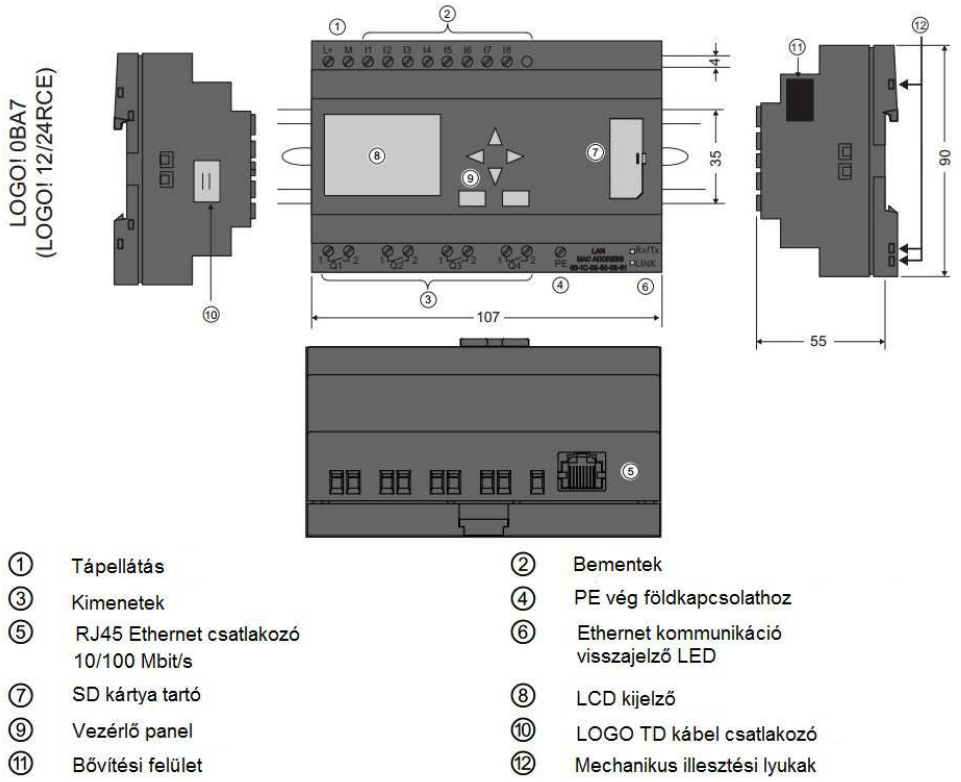
A rendszerhez szükséges hardverek és szoftverek

A projekt megvalósításához több lépésben tudtam csak eljutni. Elsőként meg kellett oldanom a jelzőfej PLC-ről való irányítását, majd meg kellett valósítanom a forgalomirányító berendezés és a PLC közötti kommunikációt. Ahhoz, hogy tisztában legyünk a rendszerben működő komponensekkel először minden felhasznált hardvert bemutatok, melyet a megvalósításnál felhasználtam. Ezek főleg a tanszéki labor eszközei, illetve egy általam készített mini jelzőfej. Továbbá ismertetem a szükséges szoftvereket is.

4.1. Siemens LOGO! 12/24 RCE és tápegysége

A LOGO! a Siemens cég univerzális logikai modulja, PLC-je, melyet a 4.1 ábra szemléltet. Képes bonyolultabb logikai feladatok megoldására és hálózati kommunikáció is kivitelezhető vele, ezért került ez a típus felhasználásra.
A LOGO! a következőket tartalmazza:

- Vezérléseket
- Kezelőpanelt és háttér világításos kijelző panelt
- Tápegységet
- Interfészt a bővítő modulok számára
- Interfészt a program modul (kártya) és a PC kábel számára – SD kártya bemenet, Ethernet csatlakozó
- Előre konfigurált alapfunkciókat, pl. be- és kikapcsolás késleltetést, impulzus relét és funkcióbillentyűt



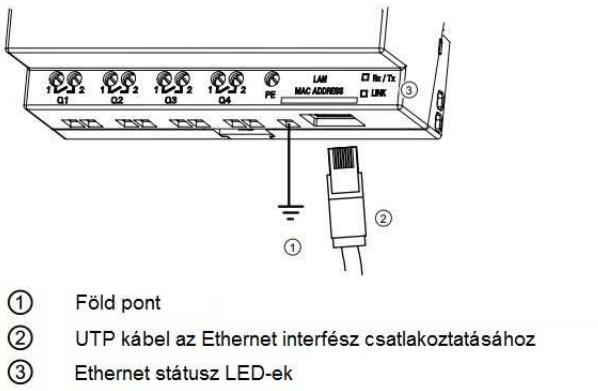
4.1. ábra. A LOGO! 12/24 RCE felépítése [Siemens, 2009]

- Időzítőt
- Digitális és analóg jelzőbiteket - visszajelző ledek
- A készülék típusának megfelelő be- és kimeneteket (8 digitális bemenettel, 4 kimenettel - relével) [Siemens, 2009]

A közúti laborban összeállított rendszeremben 3 darab LOGO!-t használtam fel. Ebből kettő két közúti jelzőfejet irányított, és a harmadik pedig egy általam készített kis mini jelzőkészüléket.

A LOGO! csatlakozói

A kivezetések fogalma a LOGO! valamennyi kimeneti és bemeneti csatlakozási pontját és azok állapotait jelenti. A bemenetek és a kimeneteke a ’0’ és ’1’ állapotokat vehetik fel. A ’0’ állapot jelenti a bemenet feszültségmentes állapotát, az ’1’ a feszültség alatti állapotot. A programozás megkönnyítésének érdekében bevezetjük a hi, lo és x állapotokat.



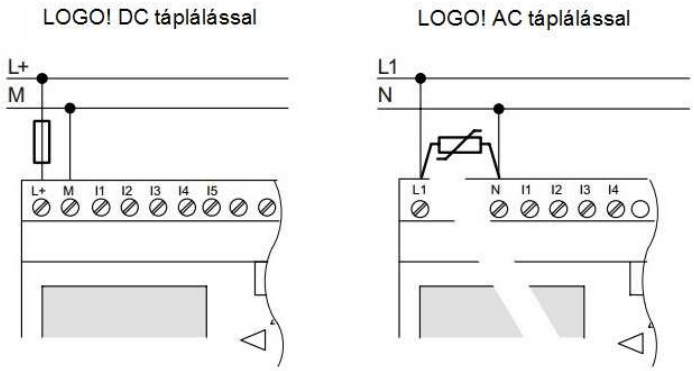
4.2. ábra. A LOGO! hálózati csatlakozói[Siemens, 2009]

- A 'hi' (high) jelenti a rögzített '1' állapotot,
- a 'lo' (low) jelenti a rögzített '0' állapotot.

A programozás során nem kell majd a LOGO! összes csatlakozóját felhasználni. A nem használt csatlakozók állapotát a program automatikusan úgy rögzíti, hogy az adott blokk megfelelő működését biztosítsa. Az 'x' el ezeket a nem használt csatlakozókat lehet jelölni.

Tápegység

A LOGO!-t egyenárammal kell táplálni 12V vagy 24V feszültségen - illetve váltóáram esetén kiegészítő ellenállás szükkséges (lásd 4.3 ábra). A jelzőfejeket irányító LOGO!-k fel vannak helyezve egy sínre, és a melléjük csatolt Siemens tápegység látja el őket.A harmadik PLC-t (ami az általam készített LED-es jelzőfejet irányította) egy univerzális, változtatható egyenfeszültséget szolgáltató tápegység látta el (12V / 2,5A).



4.3. ábra. A LOGO! tápellátása [Siemens, 2009]

A tápfeszültség bekapcsolása:
A LOGO!-nak nincs tápfeszültség kapcsolója, így feszültség alá helyezve azonnal elindul. Az, hogy mi történik a bekapcsoláskor az alábbiaktól függ:

- Van-e a LOGO!-ban tárolt program?
- Van-e behelyezett programmodul (kártya)?
- Mi volt a LOGO! bekapcsolás előtti állapota? [Siemens, 2009]

Lehetséges válaszok:

- Ha nincs program a LOGO!-ban, vagy csatlakoztatott programmodulban (kártya), akkor a „No Program / Press ESC” üzenet jelenik meg a kijelzőn.
- Ha a programmodulban van program, akkor az automatikusan átmásolódik a LOGO!-ba és az ott lévő korábbi programot felülírja.
- Ha a LOGO!-ban, vagy a programmodulban (kártyán) van tárolt program, akkor a LOGO! beáll abban a működési állapotba, amelyben a kikapcsolás előtt volt. (A kijelzőn megjelenik a dátum és az idő)
- Ha a LOGO!-nak legalább egy maradó funkcióját bekapcsolták, vagy állandó remanenciára kapcsolt funkciót alkalmaztak, akkor az aktuális értékek megőrződnek a hálózat kikapcsolása esetén is. [Siemens, 2009]

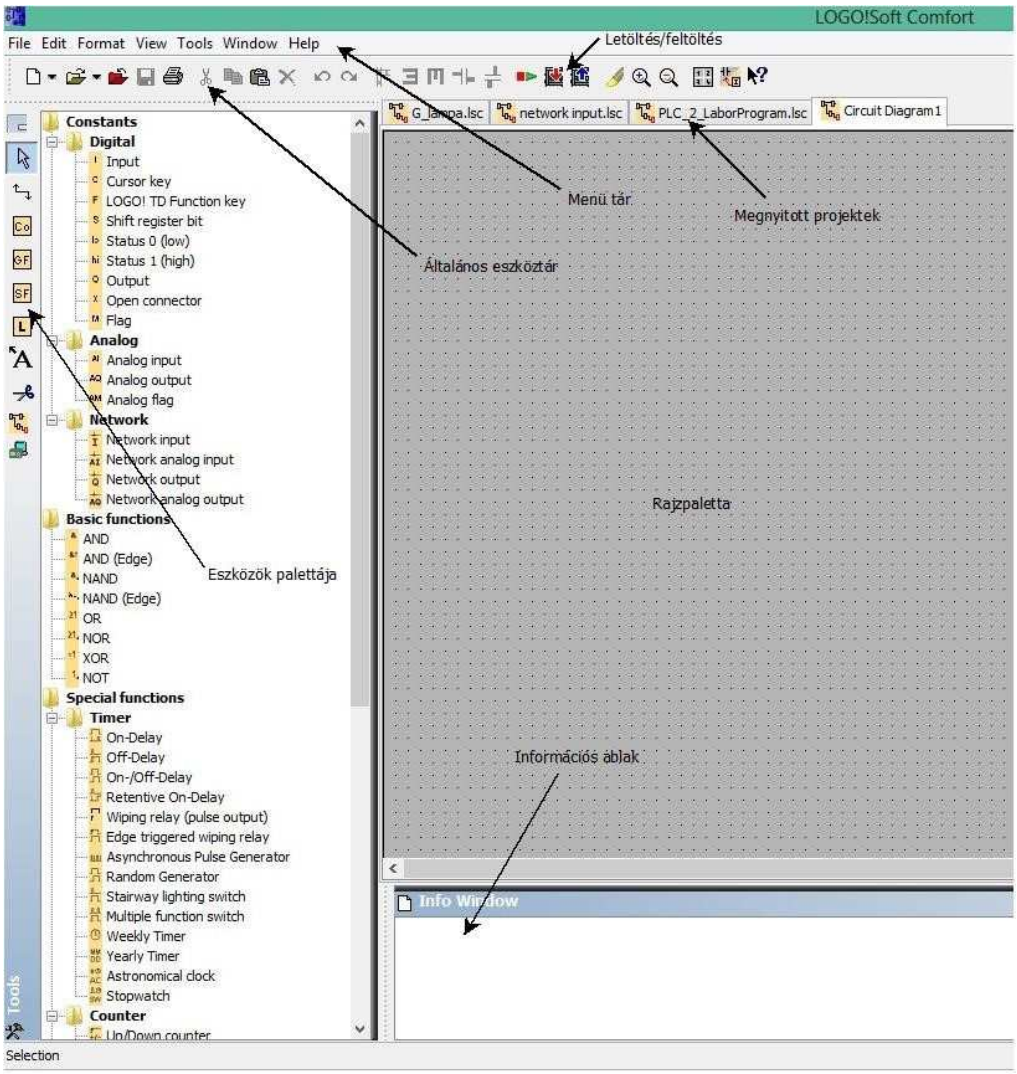
4.2. Siemens Logo Soft Comfort

A Siemens külön szoftvere a PLC-k programozásához, mely egy igencsak felhasználóbarát szoftver (lásd 4.4 ábra). Ezzel a programmal lehetőség van a vezérlőprogramok PC-n történő megírására. A program a PC-n könnyen kipróbálható, változtatható, tárolható és ki is nyomtatható. A programozáson jelen esetben valamilyen áramköri kapcsolás bevitelét értem. Egy LOGO! program tulajdonképpen nem más, mint az áramkör kapcsolási rajzainak másféle módon történő megjelenítése. A programozása nem kifejezetten nehéz, mivel a LOGO! programja előre definiált funkciójú blokkokból épül fel. A logikai hálózatot úgy kell felépíteni, hogy a bemenetek és a kimenetek közé a megfelelő blokkokat rakjuk a megfelelő kapcsolatokkal. A be- és kimenetek az alap- és különleges funkciólistában találhatók és az úgynevezett „drag and drop" módszerrel a vezérlőprogramba fűzhetők, illetve tetszés szerint összeköthetők és eltolhatók. [Siemens, 1999] A grafikus felhasználó felület nagyban hozzájárul a felhasználóbarát kezelhetőséghez. „Rajzpalettán" tudjuk létrehozni a programrészleteket és összekapcsolni őket. A be- és kimeneteket ugyanúgy, mint a funkcióblokkokat el lehet látni kommentárral, ami sok esetben megkönnyíti a program átláthatóságát. Az elkészült áramköri kapcsolást szimulálni lehet a szoftver segítségével, így könnyen felismerhetőek a hibák, problémák.
A tervezés háromféle típusban történhet:

- Funkcionális blokkdiagram
- Létra diagram
- UDF diagram

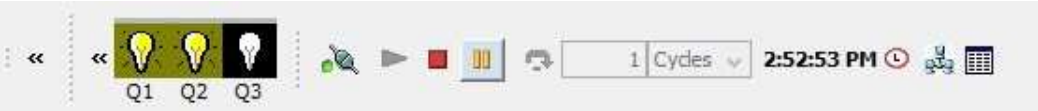
A szoftver felépítése:
A legfelső sorban érhető el a Windows-ból megszokott általános füleket. Itt lehet új projektet kezdeni, az aktuális programot módosítani, visszavonni. A Format menüben a megjelenítésre kerülő szöveget stílusait lehet állítani, a Viewban általános nézeteket. A Tools menüpont alatt lehet feltölteni az elkészült programot a LOGO!-ra. Itt lehet szimulációt illetve, online tesztet indítani, az Ethernet kapcsolat beállításait elvégezni, továbbá a VM paraméterek feltérképezni. A menü sor alatt található az általános eszköztár. Ide a leggyakrabban használt eszközök ikonjai vannak kiemelve. Ide tartozik az új projekt, a megnyitás, bezárás, mentés, nyomtatás, kivágás, másolás, törlés, visszavonás, előre lépés, szimuláció futtatása, megállítása, LOGO!-ra való fel- és letöltés. A baloldalon függőleges elhelyezéssel az eszközök palettáját található. Itt a kijelölés, összekötés, illetve a blokkok elhelyezésének funkciói vannak. A blokkok 3 csoportra vannak felosztva:

- Az első a konstansok és a konnektorok (összekötők) csoportja. Itt a legfontosabb digitális elemek az Input, Output, Status 0, Status 1, Flag. Ezek mellett itt találhatóak az analóg és a hálózati Input és Outputok.
- A második csoport az alapvető funkciókat tartalmazza. Ezek az AND, NAND, OR, NOR, XOR, NOT.
- A harmadik csoport a speciális funkciójú elemeket tartalmazza, melyek néha elengedhetetlen kellékei a tervezésnek. Ide tartoznak az időzítők, a számlálók, analóg eszközök és további vegyes eszközök.[Siemens, 1999]



4.4. ábra. A Logo Soft Comfort

A baloldali eszköz paletta mellett megjelennek az felhasználható eszközök teljes névvel, így innen könnyebben tudjuk kiválasztani az épp szükségeset.A képernyő közepén található a rajzvászon, amire készíthető az áramkör. A rajzolás történhet az úgynevezett „drag ang drop” technikával, vagy a kijelölő egér segítségével, a kívánt eszköz kiválasztása után a rajzpalettán kattintva szintén elhelyezi blokkot. A blokkok összekötése az Connect funkció kiválasztása után a blokkok széleinek összehúzásával, lenyomott egérgomb mellett történik. Az elkészült program szimulálható az F3 gyorsbillentyű segítségével. Ilyenkor a 4.5 ábrán látható eszközsor jelenik meg a rajzvászon alatt:



4.5. ábra. A Logo Soft Comfort szimulációs felülete

Itt az izzók jelentik a kimeneteket. A szimuláció elindulásával, a rajzvásznon azon kapcsolatok és elemek piros színre váltanak az általános kék színről, melyeken áram folyik. A kimenetek állapotát ezen az eszközpalettán az izzók jelzik. Ha állapotuk ’1’ akkor az izzók sárga színűre változnak és égést mutatnak. A szimuláció szüneteltethető, megállítható, illetve követhető a futásidő.

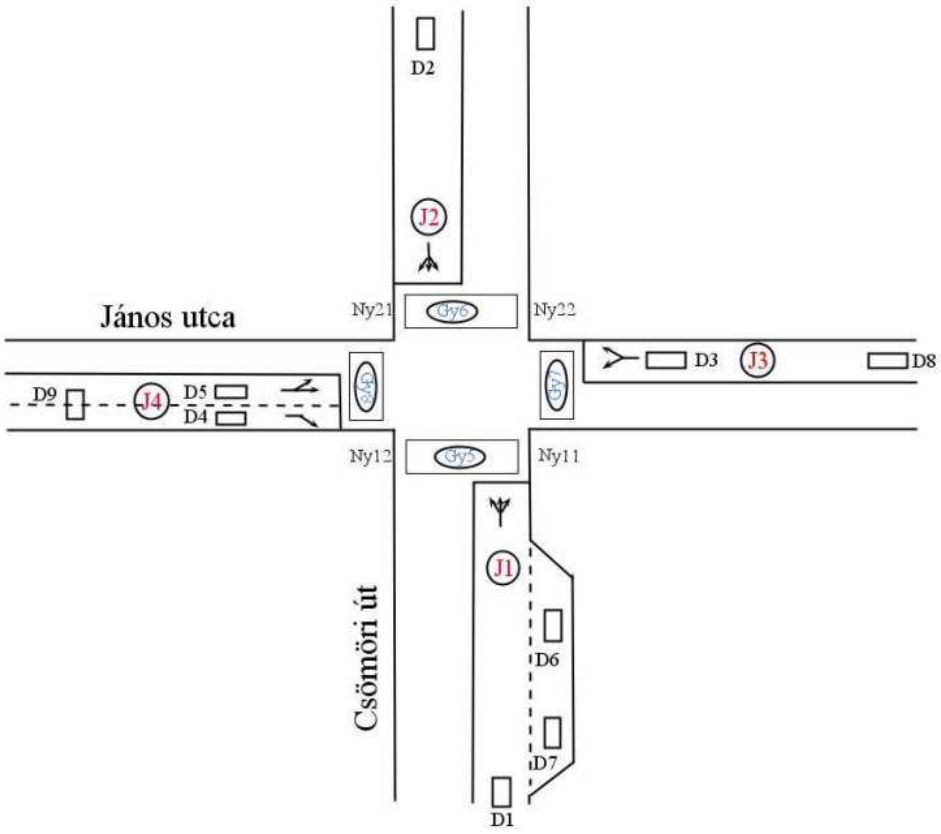
4.3. Forgalomirányító berendezés

A forgalomirányító berendezés egy Actros VTC 3000 típusú vezérlő, amely laborsegédeszköz (lásd 4.6 ábra). A berendezés - korszerű technológiájának köszönhetően - a hagyományos értelemben vett jelzőlámpás forgalomirányítási feladatok elvégzésén túl, gyakorlatilag bármilyen közúti forgalomszabályozás vezérlésére alkalmas [Tettamanti, 2010]. A forgalomirányító berendezés feladata, hogy adott csomópont jelzőfejeinek jelzésképeit vezérelje. A csomóponton működő programokat a berendezés irányítja és váltogatja az aktuális forgalomnak megfelelően. A gép egy budapesti csomópont forgalomtechnikai tervét tartalmazza (Bp. XVI kerület Csömöri út - János utca csomópont), és ennek megfelelően úgy futtatja a programokat a laborban is, mint a terepi berendezés. A forgalomtechnikai kód 4 programot tartalmaz: 3 darab fixprogramot és 1 darab forgalomfüggő logikát. Az intelligens jelzőfejjel működő rendszer esetén ebből egy darab fix jelzéstervű programot használtam fel, és a gépet minden esetben ebbe vezéreltem.

4.6. ábra. Actros VTC 3000

4.4. Eclipse

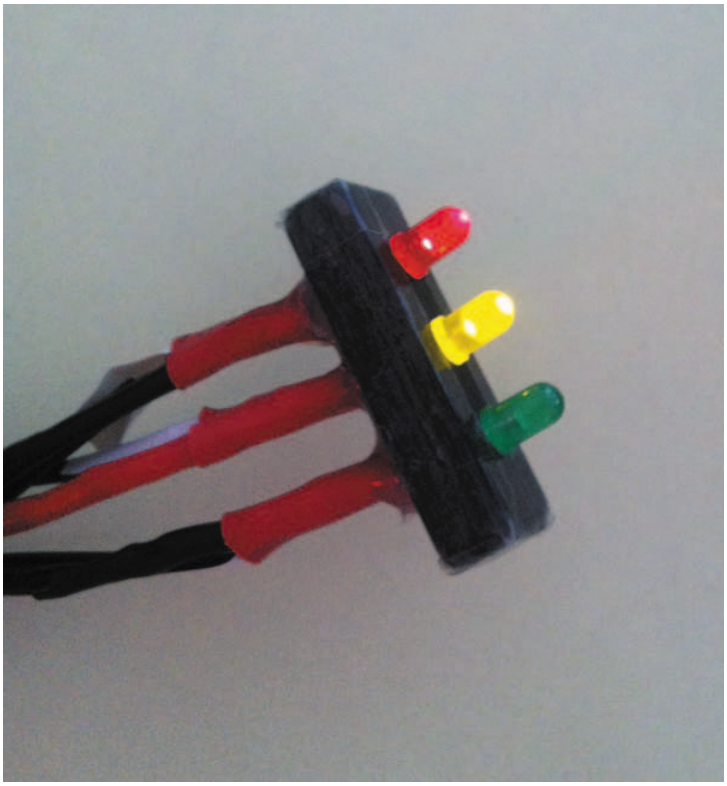
Az Eclipse egy olyan fejlesztő szoftver mely hatékonyan támogatja a Java környezetben való programozást. Működéséhez elengedhetetlen, hogy telepítsük mellé a Java Development Kit (JDK)-et is, ami egy fordító a Java SE, Java ME, és Java EE platformokra. A program felépítése nagyban hasonlít az általános programozói felületekhez. A példaprogramban a Csömöri út és a János utca csomópontjára már megírt kódokat használtam. A csomópont vázlata a 4.7 ábrán látható. Természetesen a PLC vezérelt rendszer felépítéséhez lényegében bármilyen csomópontot választhattam volna, hogy a rendszert működésbe hozzam.



4.7. ábra. Csömöri út és János utca csomópont ábrája [Tettamanti and Polgár, 2010]

4.5. Világító diódákból összeállított jelzőfej

A három diódából (piros, sárga, zöld) összeállított jelzőlámpa egy általam készített mini jelzőfej, amely megjelenítésre, ellenőrzésre alkalmas (lásd 4.8 ábra). A diódákra felforrasztottam a megfelelő előtét ellenállásokat és egy balsafa tokba ragasztottam őket, ami a lámpatestet adja. Ez az egység így beköthető a LOGO!-ba. Tápegységgel csak a PLC-t kell ellátni, mely meg fogja szakítani az áramkört (a rátöltött logika alapján). Így megkapjuk a kívánt jelzésképet. Ez az egység a labor két szabad jelzőfején kívül további lehetőséget ad a rendszerbe bevonható elemszámokhoz, ezáltal három PLC-t is fel tudtam használni.



4.8. ábra. A világító diódákból készült jelzőfej

4.6. További hardverek

- Laborban található jelzőfejek: egy darab 230V-os Elba típusú és egy korszerű 40V-os Siemens fej (mindkettő LED-es).
- Router és Hub eszközök

5. fejezet

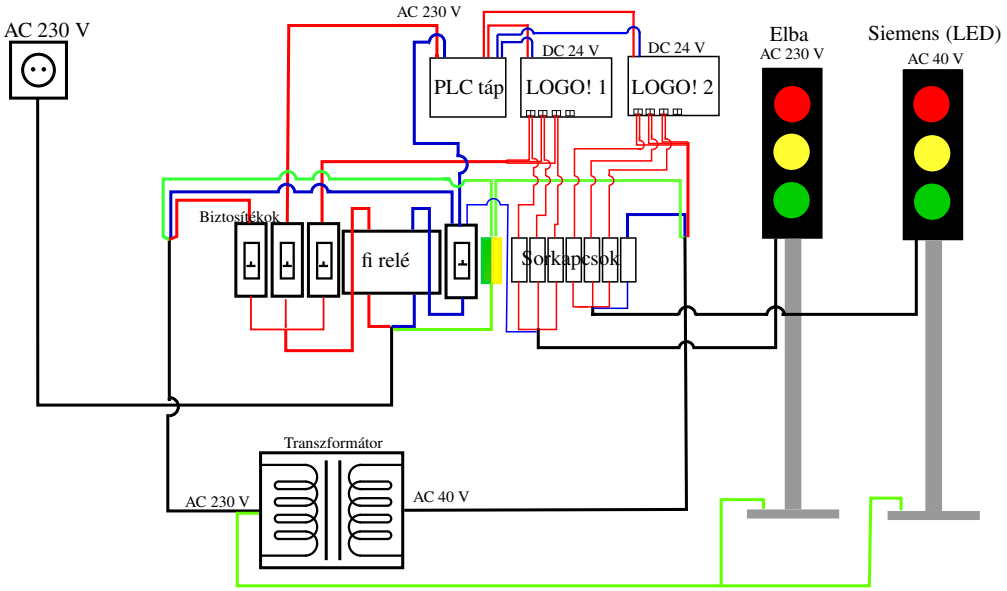
Az intelligens jelzőfejjel működő rendszer felépítése és a forgalomirányító berendezéssel való együttműködése

A következőkben bemutatom rendszerem fizikai összeállítását, melyet labor körülmények között megvalósítottam. Részletezem a rendszerem elektrotechnikai kapcsolatát és a hálózati kapcsolatok felépítését, az ismertetett hardverek összeépítésével. Továbbá bemutatom a forgalomirányító berendezés és az intelligens jelzőfej együttműködésének feltételeit.

5.1. A hardver rész összeállítása

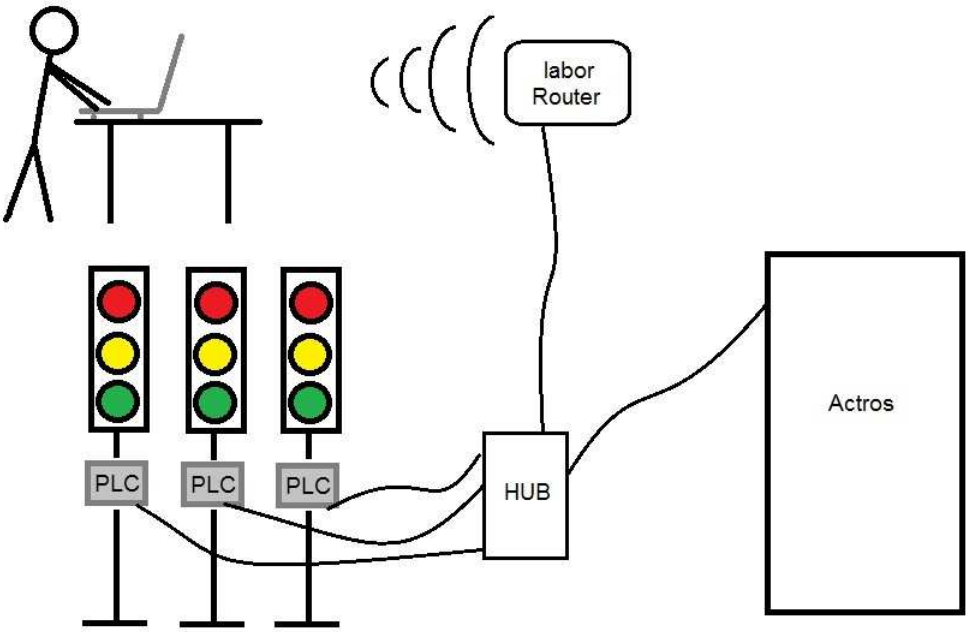
A rendszer működőképessé tételéhez az első feladatom az volt, hogy a hardver oldali eszközök tökéletesen működjenek. A komponenseket össze kellett állítanom, egymáshoz kötnöm őket. A lépéseket az 5.1 ábra szemlélteti.
Az Elba típusú jelzőfejet 230 V-on, a Siemens típusút 40 V-on kell táplálni. A hálózatról 230 V váltakozó feszültségen kaptam áramot. Ezt életvédelmi szempontból először egy fi-relébe vezettem, onnan pedig három felé vittem tovább, egy-egy biztosítékon keresztül. Elsőnek a transzformátorba, ami a 40 V-os váltóáram előállításához kellett. Másodiknak a PLC-k tápegységébe, ami 24 V-os egyenáramot állított elő belőle, így táplálva a PLC-ket. A harmadik ágat pedig az első LOGO! Output -jára, ahonnan pedig tovább az Elba jelzőfejre vezettem. A vezetékek összekapcsolásakor legtöbbször sorkapcsokat alkalmaztam. Az előbbiekben szétágaztatott 230 V-os első ágat a transzformátor primer ágára vezettem, így a szekunder ágban 40 V váltakozó feszültséget kaptam. Ezt rávezettem a máso-

dik LOGO! kimenetére, ahonnan pedig tovább a Siemens jelzőfej izzóira. A fázist mindig piros színnel jelöltem, a 0-át kékkel, a földet pedig zöld színnel.



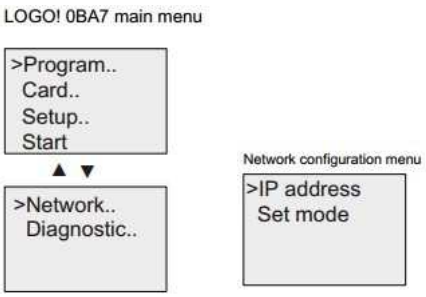
5.1. ábra. A rendszer elektrotechnikai összeállításának sematikus ábrája

A LOGO!-kat rákötöttem a tápegységre. A bemeneti oldalon az 'L+' jelzésre kötöttem a PLC tápból érkező 24 V egyenfeszültséget, az 'M' jelzésű bemenetre pedig a földpontot. A PLC-k kimeneteire rákötöttem a jelzőfejeket. Ennek menete, hogy a kimenetek első csatlakozóira kötöttem a tápláló feszültséget, majd a második csatlakozóról továbbhúztam a kábelt a jelzőfejek izzóihoz. A LOGO! lényegében a kimeneteire kötött vezetékeken képes az áramkört megszakítani és zárni, egy előre megírt program alapján.
Ezzel a PLC-k és a jelzőfejek kapcsolata létrejött és az áramellátásuk is biztosítva lett. A hálózati kapcsolat felállításához a TCP/IP protokollt alkalmaztam. A LOGO!-kat UTP kábel segítségével rákötöttem a labor hálózatára (lásd 5.2 ábra). Ez nem közvetlenül történt, ugyanis először egy Hub-ba kötöttem őket, majd a Hub-ot a labor Router-re. Az forgalomirányító berendezést is bevontam a rendszerbe. Az Actrosnak saját hálózati kártyája van Ethernet csatlakozóval, így őt szintén egy UTP kábellel be tudtam kötni a Hub-on keresztül a labor hálózatába. Ezzel a teljes rendszer közös hálózatra került. A jelzőfejek a PLC-n keresztül kommunikálni tudnak a forgalomirányító berendezéssel, illetve számítógépről programot lehet feltölteni mind az Actrosra, mind a LOGO!-ra. Az 5.2 ábrán látható egy vázlat a rendszer hálózati kapcsolatairól. A PLC-ken még be kellett állítanom a programkódban használt IP-címeket, melyek a jelezőfejek azo-



5.2. ábra. A rendszer hálózati kapcsolatainak sematikus rajza

nosítására szolgálnak. A PLC kezelő gombjaival könnyen hozzáférhetünk a LOGO! beállítási lehetőségeihez (lásd 5.3). A menüben a Network opción belül található az IP Address. Itt beállítottam, minden LOGO!-nál egy IP címet, amivel el tudtam érni őket egyesével. [Siemens, 1999]



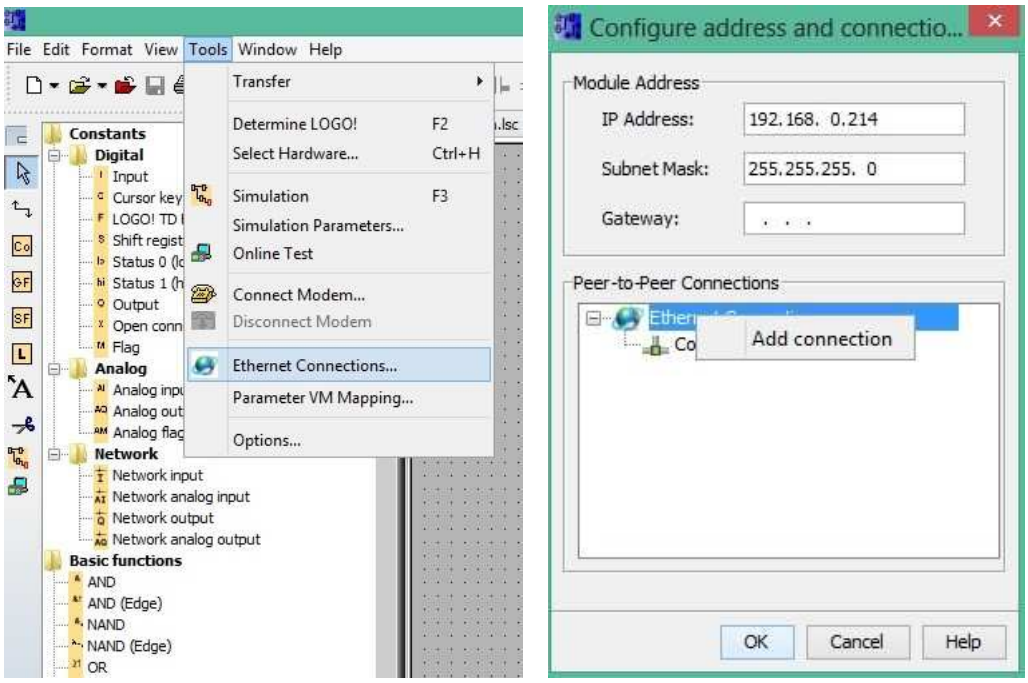
5.3. ábra. A LOGO! IP cím beállítása [Siemens, 2009]

A rendszer működőképes állapotba viteléhez elsőnként a PLC-ket kellett bekapcsolnom. A LOGO!-k elindításához nem kellett más, minthogy feszültség alá helyezzem őket. Ilyenkor a rátöltött program azonnal elindul. Ezek után indítottam csak el az Actrost. A sorrend fontos, mert a PLC-knek már „élniük” kell az Actros indulásakor.

5.2. Az intelligens jelzőfej és a forgalomirányító berendezés együttműködése

Különböző hardverek általában alapvetően nem kompatibilisek egymással. Másféle a működési struktúrájuk, más az általuk futtatható kód, így a rajtuk működő programok is eltérőek. Ez a probléma az általam felállított rendszerben is felmerült, így ahhoz, hogy a PLC-ket Eclipse alól programozni tudjam, kompatibilissé kellett tennem a LOGO!-t a Java-val.
Egy kiegészítést kellett írni az forgalomirányító berendezés programjához, amit aztán Eclipse-ben használt csomagjaimhoz hozzá importáltam. Ezt a kiegészítést egyik konzulensem Ludvig Ádám készítette, aki már sokat foglalkozott korábban ezzel a témával. Ő egy interneten elérhető a Siemens által kiadott nyílt forráskódot (prodave) dolgozott át. Ebből készítette az úgynevezett libnodave szoftverkönyvtár. Ennek felhasználásával fel tudtam állítani a kommunikációt az Eclipse és a LOGO! között, bár ennek megléte nem egyértelműen vezetett a megoldáshoz.
„A libnodave könyvtár a hivatalos Siemens PRODAVE szoftverkönyvtár kiváltására készülő, ingyenes, nyílt forrású szoftver komponens, ami a Siemens PLC eszközökkel való hálózati kommunikációt lehetővé teszi. A csomag több PLC típust és többféle kommunikációs módot támogat. A szoftverkönyvtár C nyelven íródott és aktív fejlesztés alatt áll. Többféle platformra lehet lefordítani, az elsődleges platformok Linux és Windows” [Ludvig, 2013].
A kommunikáció Etherneten keresztül történt, ezért is választottam a LOGO! 0BA7 típusát, mert ez képes hálózati kapcsolatok kezelésére. A kapcsolat felállításához Logo Soft Comfortban további hálózati beállításokat kellett elvégeznem. A következőkben ezeket foglalom össze Ludvig [2013] munkája alapján, melyek a libnodave könyvtár használatához szükségesek:

- A Logo Soft Comfortban a hálózati tulajdonságok beállítását a Tools menü, „Ethernet Connections...” fül alatt lehet elérni (lásd 5.4 ábra). Itt be kellett állítanom az IP-címet, és a „Subnet Maskot” (magától kitölti). Ez mindig az az IP cím volt, amelyik PLC-vel való kapcsolatot konfiguráltam.



5.4. ábra. Logo Soft Comfort hálózati tulajdonságainak elérési útvonala [Ludvig, 2013]

- Ezek után az alattuk lévő ablakban az „Ethernet connections” szövegre jobb gombbal kattintva „Add connection”-t választottam (lásd 5.4 ábra), amivel új kapcsolatot tudtam létrehozni. Ezt a saját feladatomnak megfelelően tudtam konfigurálni. Majd a „Connection1(Server)” –re kettőt kattintva lehet a tulajdonságait elérni.

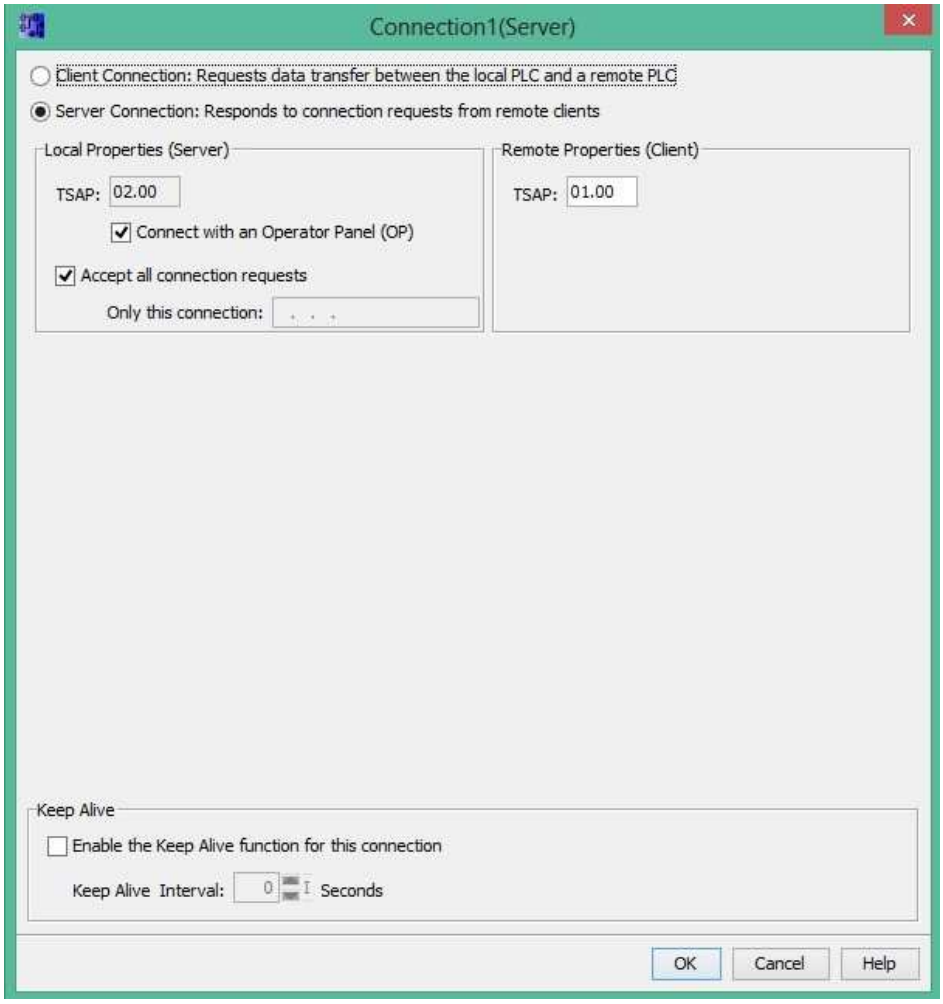
- A „Server Connection” típust választottam a rádió gombok közül (lásd 5.5 ábra). Ekkor a LOGO szerverként viselkedik és fogadja a hálózati csatlakozásokat. Kliens kapcsolatnál egy másik Siemens eszköz IP-címét és TSAP[1] -ját beállítva, annak a VM[2] -jéből kérdezhetőek le értékek.

- Bekapcsoltam a „Connect with an Operator Panel (OP)” funkciót

- Bekapcsoltam az „Accept all connection request”-et. Így minden IP címről elfogadta a csatlakozási kéréseket a LOGO!. Ez opcionális, ha korlátozni szeretném a LOGO! elérhetőségét, akkor itt van rá lehetőség.

[1] Egy csatorna azonosító szám, amivel egy eszközön egy kommunikációs csatorna kijelölhető, olyan, mint a TCP/IP kommunikációban a port szám, de ez eggyel magasabb OSI rétegben van. Bármire állít-ható két Siemens eszköz között.

[2] Variable Memory, a PLC-ken kialakítható egy, közös változóknak fenntartott memória rész, amiből lehet lekérdezni adatokat és írni rá adatokat. Ezen adatok jellemzője a cím (eltolás byte-ban) és a hossz (byte-ban).



5.5. ábra. A kapcsolat beállításai [Ludvig, 2013]

- A „Remote properties (Client)” keret alatt a csatlakozó, kliens eszköz TSAP címét átállítottam 01.00-ra, mert a libnodave csak ezzel működik.
- A „Keep Alive” (életjel) funkcionalitást kikapcsoltam, mert ezt nem támogatja a libnodave könyvtár még.

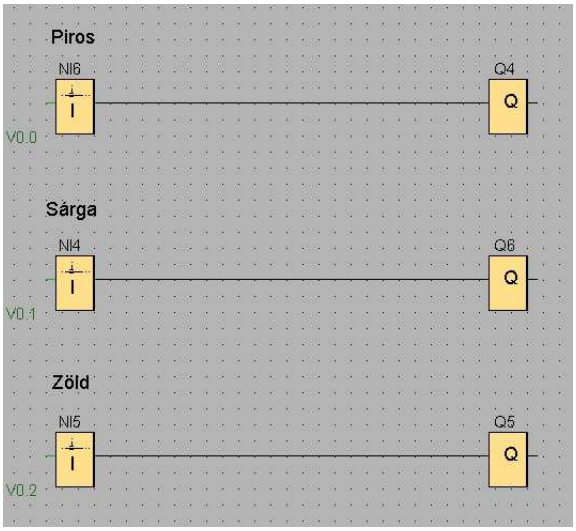
6. fejezet

Programalkotás a berendezésekre

A következőkben bemutatom a PLC-n (Logo Soft Comfortban tervezett) és az Actroson (Eclipse szoftverrel programozott) futó programokat. Ezek tervezési lépéseit és program elemeit ismertetem, továbbá a kialakításuk részleteit. Az első rész könnyebb megértéséhez a Mellékletben (A) egy egyszerű példa található.

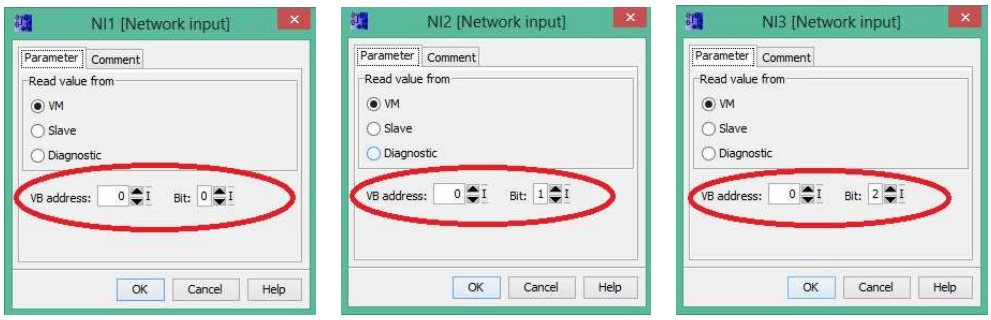
6.1. A PLC-n futó program tervezése

A LOGO! és a Java együttműködéséhez, illetve az Eclipse alól történő programozásához elegendő az alábbi (lásd 6.1 ábra) egyszerű program is, ami 3 darab Network Input -al és 3 darab Output -al rendelkezik.



6.1. ábra. A kapcsolathoz szükséges blokkdiagram

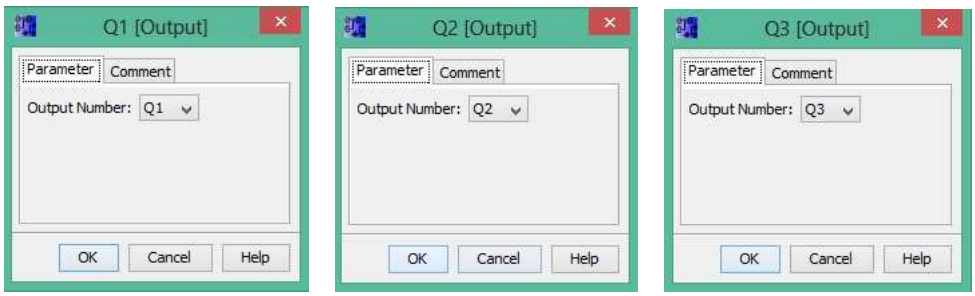
Az Actrosról való irányítás esetén, a LOGO!-ra nem tölthetek fel egy olyan fix programot, ami egy adott jelzéstervet tartalmaz, mert akkor azt csak úgy tudnám változtatni, hogy a meglévő programot mindig törlöm a PLC-ről, majd készítek egy másik jelzéstervet tartalmazó programot Logo Soft Comfortban és azt a letörölt helyére visszatöltöm. Ez a művelet elég bonyolult, és nem enged közvetlen hozzáférési lehetőséget a jelzésterv változtatására. Ezért a LOGO!-ra első lépésben egy olyan programot terveztem, ami megteremti a közvetlen és folyamatos kapcsolatot az Actrossal, így alkalmas változó jelzéstervek kezelésére. Ezen blokk diagrammal a LOGO! tudásának csak egy szerény részét használtam ki, csak azon funkcióit, amivel a feladatokat elvégzi, azaz nyitja és zárja az áramköröket. Ez a program a 6.1 ábrán látható.
A lámpa három színe miatt három bemenetre volt szükségem, azok közül is a hálózati tulajdonságok kezelésére alkalmasakra. Ezeket a Network Input -okat elhelyeztem a rajzvásznon, majd bal egérgombbal duplát kattintva, eljutva a tulajdonságai közé beállítottam a VB address-ét (lásd 6.2 ábra). Ez azt mondja meg, hogy a LOGO! memóriájának, melyik címén dolgozik illetve, hogy azon belül melyik bitet kezeli.



6.2. ábra. Network Inputok konfigurációja

Az adatok a LOGO! VM tárolójába írhatóak, és abból később könnyedén ki is olvashatóak. Ez nagyon fontos, mert az Eclipse-ből így tudtam a megfelelő izzóra kiadni az utasítást.
„A VM egyes byte-jaihoz a PLC program elemeinek tulajdonságait lehet kötni. Ezáltal ezeket a helyeket a VM-ben a PLC minden frissítési ciklusban feltölti az aktuális értékekkel” [Ludvig, 2013].
A Tools menü alatt található „Parameter VM mapping” ablakban lehet beállítani, hogy melyik blokk melyik tulajdonságát a VM-en belül hova írja ciklusonként a LOGO!. A Block és a Parameter oszlopokban lehet kiválasztani a kívánt függvényblokkot és annak kívánt paraméterét. A típusa adott a Type oszlopban és az Address alatt beállítható a VM-en belüli címe. „Fontos az, hogy nem ellenőrzi a Logo Soft Controll, hogy diszjunkt címeket adtunk-e meg, ezért tudnak ütközni

és átfedni is, ekkor egyik érték felülírja a másikat” [Ludvig, 2013]. Új sort a helyi menüben lehet hozzáadni jobb kattintással.
A programom további elemei a három darab Output. Ezeket egyszerűen csak össze kellett húznom egy-egy ’Network Input’-al, így a program elkészítette köztük a kapcsolatot. A kimenetek a jelzőfej lámpáit szimbolizálják. A Kimenetek Block Properties -énél látható, illetve módosítható, hogy ennek mi a sorszáma. Ez adja meg, hogy melyik kimenet, melyik izzóhoz fog tartozni. Ezeket beállítottam sorban 1-től 3-ig (lásd 6.3 ábra).



6.3. ábra. Az Outputok tulajdonságai

A jelzőfej izzóit a PLC- megfelelő fizikai kimeneteire kellett kötöm, illetve az előbb beállított sorrendnek megfelelően, így a helyes képet kaptam a program futtatásakor. Ezzel nem fordulhatott elő olyan hiba, hogyha például a piros színt akartam felkapcsolni, akkor a zöld villant fel. A kész programmal már tudtam működtetni a rendszert Eclipse alól. A blokk diagram később tovább lett fejlesztve, hogy több funkció ellátására is alkalmas legyen. A programot fel kellett tölteni a LOGO!-kra.

6.2. Az Eclipse és a program elemei

A következőkben az Eclipse-ben készített program részekről és az eredeti programon végzett átalakításokról fogok írni. Az Actroson található Csömöri út és János utca csomópontra megírt program java fájljai jó alappal szolgálta a LOGO!-s projektem elkészítésében. A PLC-k bevonása miatt változásokat kellett végrehajtani az eredeti kódon, illetve néhány új résszel kiegészíteni.
Az Eclipse-ben végrehajtandó feladatokkal lépésenként haladtam. Elsőként nyitottam egy új projektet. Ezt a File – New – Project fül alatt találtam. A felugró ablakból a Java Pro-ject-et választottam, majd be kellett állítanom a mentés helyét. Ezzel létrejött a projektem. Következő lépésben beimportáltam a szükséges Package-eket és fájlokat. Ezek közé tartozott az 5.2. fejezetben említett libnodave szoftverkönyvtár, illetve a Ludvig Ádám által elkészített logo.java fájl is, ami a

LOGO! hálózati kapcsolatának felállításához elengedhetetlen. Az importálási lehetőséget a program bal oldalán elhelyezkedő Package Explorer részben, a jobb egér gombbal kattintásával értem el. A felugró ablakban kiválasztottam a behozandó fájl típusát, majd végig kellett mennem az importálással kapcsolatos beállításokon. Ezek után ezt a projektet exportáltam egy JAR fájlba, ami a kapcsolat felépítéséhez kellett később. Ezt szintén a Package Explorer -ből végeztem.
Ezek után nyitottam egy új projektet, amiben már a tényleges munkámat végeztem. Elsőként az előzőekben kimentet JAR file-t elérési útvonalát kellett beállítanom. Ezt a Projects – Properties fül alatt, majd a felugró ablakban a Java Build Path – Libaries – Add external JAR... menüpont alatt tudtam elvégezni. Ezzel a kapcsolatot az Eclipse oldaláról elkészítettem. A Logo Soft Comfortban is be kellett állítanom a kapcsolat tulajdonságait, amit már az 5.2. fejezetben bemutattam. Ennek megléte esetén, a kapcsolatot a LOGO! oldaláról is késznek tekinthettem, így elkezdhettem programozni Eclipse-ben.
A következőkben a projektem java fájljait fogom bemutatni, azok fontosságának részletességével.

6.2.1. Var.java

A Var, azaz Variable, magyarul változók rövidítése. Ebben a programrészben az általánosan, a program minden részében felhasznált változókat kell definiálni. Ezek a változók public, azaz publikus, mindenki számára elérhetőek, és static, azaz statikus - értéküket nem változtatják - tulajdonságúak. Itt az összes jelzőcsoportot, ismétlőt, detektort, programot, lámpatípust, mint osztályokat definiálni kell. Nekem is definiálnom kellett a LOGO!-inkat egy új osztályként.

```
public class Var
{
    public static LogoJelzofej logo;
    public static LogoJelzofej logo1;
    public static LogoJelzofej logo2;
}
```

Mindhárom LOGO!-t publikus és statikus osztályként definiáltam LogoJelzofej névvel. A Var osztály az Actros indulásakor egyszer fut le, amikor a deklarálás megtörténik.

6.2.2. Init.java

Ebben a programrészben kell inicializálni, azaz kezdő értéket adni, minden működéssel kapcsolatos dolognak. Ezek közé tartoznak:

- jelző csoportok (Signal group)

```
private void initialisiereSg()
```

- közbenső idő mátrix (Zwischenzeitmatrix)

```
private void initialisiereZwz()
```

- detektorok

```
protected void initialisiereDet()
```

- programok

```
protected void initialisiereProgs()
```

- óra

```
protected void initialisiereUhr()
```

A programozás során mindent hagytam a meglévő formájában, csak a jelző csoport részt kiegészítettem a Logo-k inicializálásával. Ezt a következőképp tettem:

```
private void initialisiereSg()
    {
        Var.logo = new LogoJelzofej("192.168.0.212");
        Var.logo1 = new LogoJelzofej("192.168.0.213");
        Var.logo2 = new LogoJelzofej("192.168.0.214");

        System.out.println("Jelzocsoportok letrehozva");
    }
```

Ezzel az előzőekben a változóknál létrehozott LogoJelzofej osztályokat inicializáltam. A LOGO!-khoz, amelyik *logo, logo1, logo2* névvel futnak, hozzá rendeltem az IP címeiket. A definiálást a *LogoJelzofej.java*-ban adtam meg a következő módon:

```
logo = new Logo (address);
```

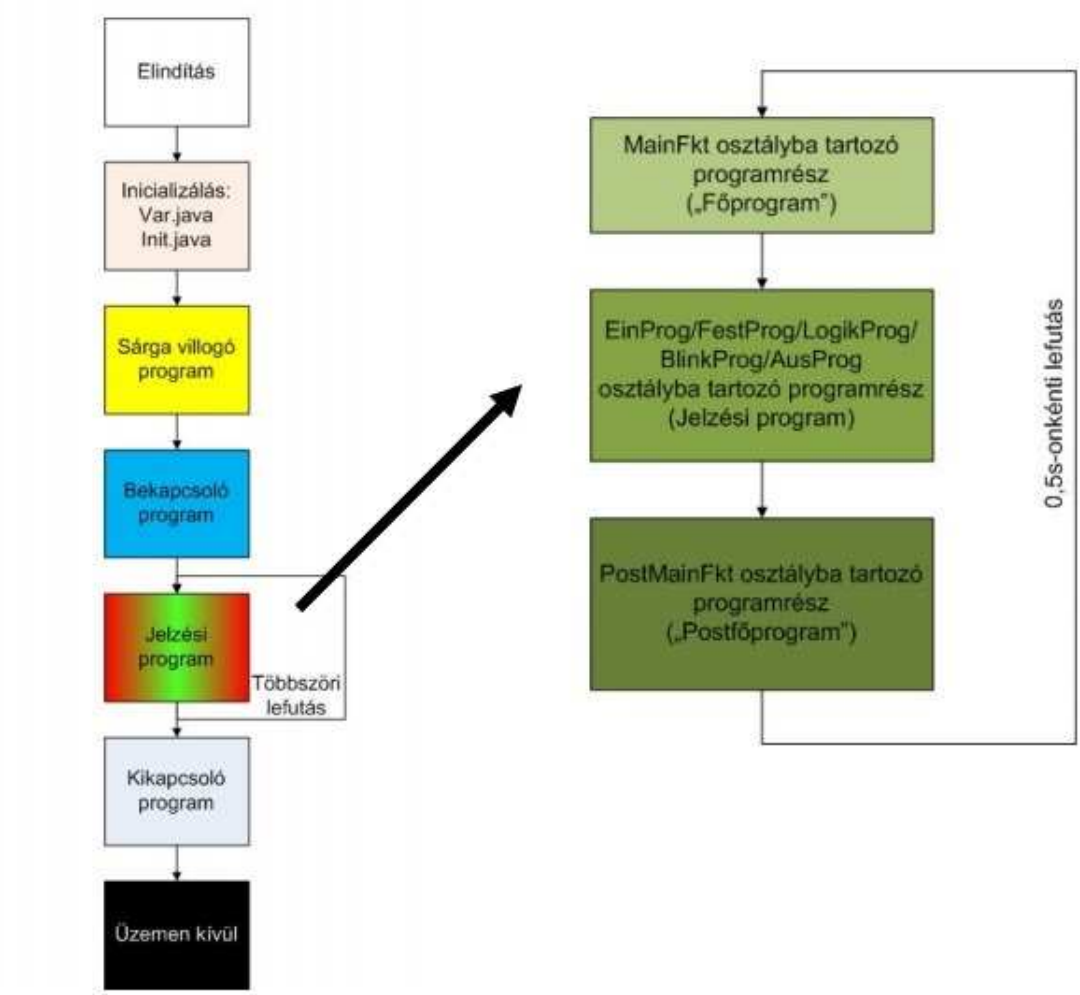
Innentől kezdve, ha például a *logo1* –re hivatkozok, az a .213 IP cím végű PLC-re lesz érvényes.

6.2.3. BeProg.java

Az osztály feladata a „beprogramozás”. Egy csomópontra tervezett program nem indulhat el azonnal a bekapcsoláskor, így mindig alkalmaznak egy belépő programot. Ez segíti a különböző fázisok megfelelő időben való elindulását, úgymond rávezeti a programokat az elindított idősávra.
A LOGO!-s projektemben ezt a programrészt nem nagyon kellett átalakítanom, ugyanis én az egyik meglévő fix programba építettem bele a LOGO!-s programozást. Az megfelelő, ha a BeProg elindítja a fix programot. Egy rövid kiegészítést írtam, ami a BeProg programrész futásakor az összes intelligens jelzőfejet piros jelzésre állítja. Így elinduláskor az alap állapot az, hogy minden jelző tiltást mutat.

6.2.4. AltalanosResz.java

Az Actros elindulása, azaz a változók definiálása, inicializálás és a bekapcsoló program lefutása után belép a program a jelzési programba. A jelzési program az, amiben egy adott csomópontra tervezett jelzésterv le van programozva. Ez a programrész minden fél másodpercben lefut. Az Actroson működő programban ennek része az AltalanosResz.java a (6.4 ábrán ez MainFkt néven fut). Ebben az osztályban hívtam meg a Var.prog1 –et, ami úgy van inicializálva, hogy a FixProg1-re mutat. A működést a 6.4 ábra szemlélteti.



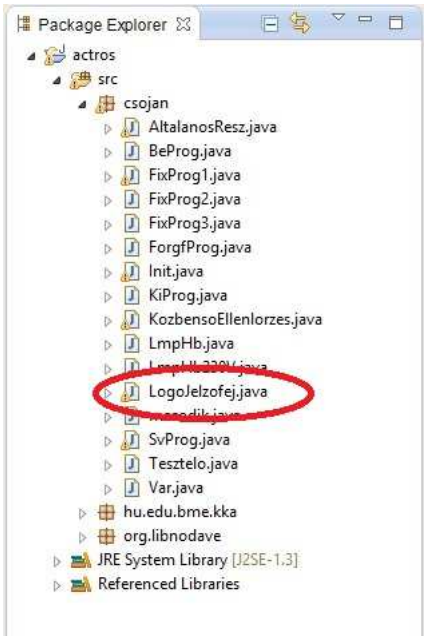
6.4. ábra. Az Actros működése [Tettamanti and Polgár, 2010]

6.2.5. LogoJelzofej.java

Ezt az osztályt én készítettem a Logo-val kapcsolatos működések leprogramozására. Új osztályt az Eclipse-ben a File – New – Class menüpont alatt lehet készíteni. A felugró ablakban csak a nevét kell megadni, és a program máris létrehozza az osztályt. Az osztály a 'csojan' package-be fog tartozni (lásd 6.5 ábra), azaz a Csömöri út – János utca csomópontra írt programhoz. A 'csojan' ennek a rövidítése. Az osztály tetején az importált részekben a kivétel kezeléshez kapcsolatos részek, illetve a logo-s szoftverkönyvtárunk van. Ezeket mindenképp importálni kell, hogy a kapcsolat a PLC-kkel működjön. Ezt az alábbi kóddal tettem:

```
package csojan;

        import java.io.IOException;
        import java.net.UnknownHostException;
        import vt.StgEbene;
        import hu.edu.bme.kka.Logo;
```



6.5. ábra. Package Explorer

Ezek után a public class utasítással indul az osztály. Elsőként meghívtam a Logokat. Ezt lényegében a *logo1 = new Logo (address);* kóddal tettem. Itt a logok-hoz, melyek *logo, logo1, logo2* névvel futnak, hozzárendeltem IP címek alapján egy-egy PLC-t.

Try-catch hibakezelés

A try catch formula egy eljárás a hiba kezelésre. A try paranccsal a programunk egyes részeihez hibakezelő rutinokat lehet rendelni, és ezek hiba esetén az előre megírt hibakezelő eljárásokat hajtják végre, illetve megakadályozzák a program leállását. Az Eclipse hibakezelés nélkül nem is engedi lefutni a kódokat. A catch segítségével el lehet kapni azokat a kivételeket, melyek a try blokkban keletkeztek. Itt meghatározható, hogy hogyan kezelje az eljárás a felmerülő hibákat. A

try-catch formula azért került felhasználásra, mert ez az egyik legegyszerűbb hibakezelő rutin. Az alábbiakban látható a megírt kód:

```
public class LogoJelzofej {
	private Logo logo1;
	public LogoJelzofej(String address) {
		try {
			logo1 = new Logo (address);
		} catch (UnknownHostException e) {
			e.printStackTrace();

		} catch (IOException e) {
			e.printStackTrace();

		}
	}
```

BEKI függvény

A következőkben egy függvény magyarázata következik, amit ebben az osztályban készítettem el. Ez az úgynevezett BEKI függvény, melynek neve a belépő, és kilépő idő rövidítéséből ered. A BEKI a 'csojan' mappában már egy létező függvény volt. Arra szolgált, hogy adott jelzőt vezéreljen. A paraméterében meg lehetett adni a jelző nevét, a zöld kezdetének és végének az idejét. Nagyon praktikus és elegáns függvény, mert nem kódban kellett leprogramozni, hogy mettől meddig tartson a zöld idő, hanem ezt a feladatot a háttérben futó függvény végezte el, ami mindig meghívásra került. Ez így nézett ki az eredeti programban:

```
	K.BEKI(Var.j11, 1, 28);
```

Ennek jelentése, hogy a j1 irány első jelzője az első másodperctől a huszonnyolcadik másodpercig legyen zöld. Egy ilyen függvényt készítettem a LOGO!-hoz. A nevét megtartottam, meghívása teljesen hasonlóan történt, mint az eredeti BEKI függvénynek a FixProg.java –ból. A BEKI utasítás után a zárójelben lévő (kezdési idő, befejezési idő, logo száma) bármilyen értéket felvehet, mely a ciklusidőn belül van, illetve a harmadik paraméter azt adja vissza, hogy melyik PLC végzi a parancsot. A meghívás teljesen hasonlóan történik az én függvényemben is, mint az eredetiben:

```
	Var.logo1.BEKI(25, 40);
```

Az, hogy melyik PLC-re vonatkozzon, azt a *logo* utáni szám határozza meg. A kód azt jelenti, hogy a logo1-hez tartozó IP című LOGO! a 25. másodperctől a 40. másodpercig legyen zöld. Ezt a függvényt a LogoJelzofej.java-ban készítettem el. Ennek magyarázata következik. Az egyes lámpa állapotokat definiáltam először. Ezeket külön változókba vettem fel, hogy a kód szebb legyen, és értékül adtam nekik, a megfelelő idő intervallumokat, illetve címeket:

```
private static final int PIROSSARGA_IDO = 2;
    private static final int SARGA_IDO = 3;
    private static final byte PIROS = 1;
    private static final byte SARGA = 2;
    private static final byte ZOLD = 4;
    private static final byte PIROSSARGA = PIROS|SARGA;
```

A PIROSSARGA IDO és a SARGA IDO változó értékei időt jelentenek másodpercben, a többi változó azt az értéket, amit a PLC-re kell írni, hogy adott égő világítson. Ezek után a kezdő és befejező pillanatokat definiáltam. A getBefejezes() függvény visszatérési értékét a 'bef' változóba írattam, a getKezdes() függvény visszatérési értékét pedig a 'kezd' változóba. Ezek a függvények, melyek a 'csojan' részei, a kezdő és befejező időt adták meg. Ezt így készítettem:

```
private int bef;
    public int getBefejezes() {
                return bef;
        }

private int kezd;
        public int getKezdes() {
                return kezd;
        }
```

Majd létrehoztam a BEKI függvényt. Public void függvényként definiálva, azaz publikus és nincs visszatérési értéke. Három paramétere a 'kezdes' és 'befejezes' és 'logoszama' változók lettek, melyekből az első kettőt egyenlővé tettem, az előbb létrehozott 'kezd' és 'bef' változókkal. Ezek lesznek a zöld idő kezdő és befejező értéke. A 'logoszama' változó visszaadja, hogy épp melyik PLC végezte a feladatot. Ezek után definiáltam a 'sec' változót, mely az aktuális időt adta meg. Ezt egyenlővé tettem a getProgsek() függvénnyel, ami megadja a program aktuális szekundumát. Ezen kívül definiáltam még a ciklusidőt is egy 'ciklusido' változóba ugyan ezzel a metódussal, csak ott a getZyklDauer() függvényt használtam. Ezek a függvények szintén a 'csojan' részei.

```
public void BEKI(int kezdes,int befejezes,final int logoszama) {
              int sec = Var.tk1.getProgSek();
              int ciklusido = Var.tk1.getZyklDauer();
              bef = befejezes;
              kezd = kezdes;
```

A BEKI függvény lényegében 'if' állításokból áll. Feltételek felállításával építettem fel a függvényt úgy, hogy teljesen általános legyen. Így bármelyik jelzőnél fel tudtam később használni, hiszen csak a paramétereit kellett megadnom. A kódsor itt látható:

```
              if (sec == befejezes + SARGA_IDO){
                            set(PIROS, logoszama);
                            }
              if (sec >= kezdes && sec < kezdes + PIROSSARGA_IDO){
                            set(PIROSSARGA, logoszama);
                            }
              if (sec == kezdes + PIROSSARGA_IDO){
                            set(ZOLD, logoszama);
                            }
              if (sec >= befejezes && sec < befejezes + SARGA_IDO){
                            set(SARGA, logoszama);
                            }
```

A kód magyarázata:

- **Piros szín**
  Ha az aktuális szekundum egyenlő volt a 'befejezes' változó és a sárga idő összegével, akkor kiadtam a LOGO!-nak a piros égést. Ezt úgy értem el, hogy a megfelelő bitre adtam igaz értéket. Ezek egyezését már beállítottam a definiáláskor, illetve a Logo Soft Comfortban. A piros színhez például az 1. bitet kellett felülírnom (fent definiáltam: *private static final byte ZOLD = 1;*).
  A set egy függvény, mely beállít a LOGO!-nak adott bitjére adott értéket. Erről a függvényről később fogok írni, ezt is én készítettem, abból a célból, hogy a kódolás tisztább és átláthatóbb legyen.

- **Piros-sárga szín**
  Ha az aktuális szekundum nagyobb volt vagy egyenlő, mint a 'kezdes' változó, valamint kisebb, mint a 'kezdes' változó és a piros-sárga idő összege (ezt

2 másodpercre definiáltam:*private static final int PIROSSARGA IDO = 2;* mert Magyarországon 2 másodpercig ég együtt a piros a sárgával zöldre váltás előtt), akkor kiadtam a LOGO!-nak a piros-sárga együtt égést a set függvény meghívásával.

- **Zöld szín**
  Ha az aktuális szekundum egyenlő volt a 'kezdes' változó és a piros-sárga idő (2s) összegével, akkor beállítottam a LOGO!-nak a zöld égést.

- **Sárga szín**
  Ha az aktuális szekundum nagyobb vagy egyenlő volt, mint a 'befejezes' változó valamint kisebb, mint a 'befejezes' változó és a sárga idő összege (ezt 3 másod-percre definiáltam:*private static final int SARGA IDO = 3;* mert Magyarországon 3 másodpercet ég a sárga izzó, pirosra váltás előtt), akkor kiadtam a LOGO!-nak a sárga égést.

A set függvény

A set függvény szolgált arra, hogy a LOGO!-nak ki lehessen adni egy adott utasítást. A LOGO! VM-ére (Variable Memory) írni az alábbi függvényekkel lehet:

```
Logo.setByte(int pos, int value)
Logo.setWord(int pos, int value)
Logo.setDWord(int pos, int value)
```

A zárójelben két paramétert kell megadni vesszővel elválasztva, az első a pozíció, a második az érték, amit a helyre szeretnénk írni. Lehet bájtot (Byte), szót (Word) és dupla szót (Dword) írni a VM-be. A függvény visszatérési értéke az aktuális Logo példány, ha halmozni szeretnénk az írást.
Problémát jelentett a kommunikáció sebessége. Az Actros úgy működik, hogy minden másodpercen kétszer lefuttatja a kódokat, tehát 0,5 s-ként lefutott a LogoJelzofej.java, ami használta a set függvényt. Ebbe a fél másodperbe bele kellett volna férjen, hogy mind a három PLC-hez eljusson az utasítás és azok végrehajtsák. Az utasítások egymás után lettek kiküldve, amire a rendelkezésre álló idő nem volt elegendő. Így az Actros megállt mikor a programnak el kellett volna indulnia, mert az adott sebesség alatt nem zajlott le mind a három LOGO! kommunikációja. A probléma megoldására a szálakra bontás vezettem be. A main szálon kívül további két szálat indítottam el, melyek már ellátták a szükséges feladatokat. A set függvényt a LogoJelzofej osztályban írtam meg, mert itt használom a függvényt. Privát függvényként definiáltam, void-nak azaz visszatérési érték nélkülinek. A set függvény kódja itt látható:

```
    private void set(final int szin, final int logosz){ //Runnable
        interface implementalasa
    class Szal implements Runnable{
                    private int szin_ = szin;
                    private String nevem;
                    public Szal(String _nevem){nevem = _nevem;}
                    public void run(){
                                    try {
                                            Thread.sleep(5);
                                            logo_i.setByte(0,szin_);}
                                    catch (InterruptedException e){
                                            e.printStackTrace();
                                            System.out.println("Szal
                                                hiba");
                                    System.out.println(e.getMessage());}
                                                                    }
            }
            Szal s0 = new Szal("Szal 00"); //Az altalam keszitett
                osztaly, ami implementalja a Runnable feluletet
            Thread sz0 = new Thread(s0); //A futo szal letrehozasa,
                es atadom neki az elobb keszitett osztalyt
            sz0.start();  //A szal inditasa
    }
```

A kódban a Runnable interface implementálásával készítettem új szálat. A szálak későbbi azonosításának céljából elneveztem. A szálkezelésnél is szükség volt try-catch hibakezelésre.

6.2.6. FixProg1.java

A FixProg1.java a 'csojan' (Csömöri út – János utca kereszteződés) egy alap fix jelzéstervű programját tartalmazza. A rendszer működésének szempontjából nem meghatározó, hogy melyik programból van meghívva a Logo-s BEKI függvény. Így az első fix programból, azaz a 'FixProg1'-ből hívtam meg. A 'csojan'-nak még van 'FixProg2', 'FixProg3' fix jelzéstervű programjai és egy 'ForgfProg' nevű forgalomfüggő programja. Az AltalanosResz.java –ban azt állítottam be, hogy a 'FixProg1.java' program induljon el. Ezek után már csak meg kellett hívnom a FixProg1.java –ban az előzőekben a LogoJelzofej.java –ban megírt BEKI függvényem. A FixProg1-ben lévő kód így néz ki:

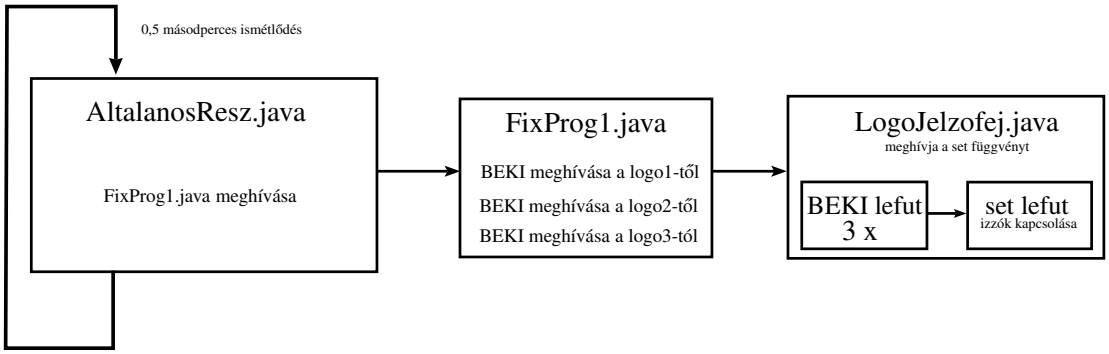
```
public void programmFunktion()
    {
        Var.logo.BEKI(0, 20, 1);
        Var.logo1.BEKI(25, 40, 2);
        Var.logo2.BEKI(45, 55, 3);
    }
```

A BEKI-t publikus függvényként definiáltam, így tudtam másik osztályban is használni. A Var.logo.BEKI (x, y, z) meghívta a függvényemet. A 'FixProg1.java' is minden fél másodpercben lefut, így 0,5 másodpercenként a program átugrott a LogoJelzofej.java –ban megírt kódrészre és lefutott az ott megírt logikai feltétel. A zöldidő hosszát teljesen egyszerűen az 'x' és 'y' helyére behelyettesített értékek közötti idő adta.

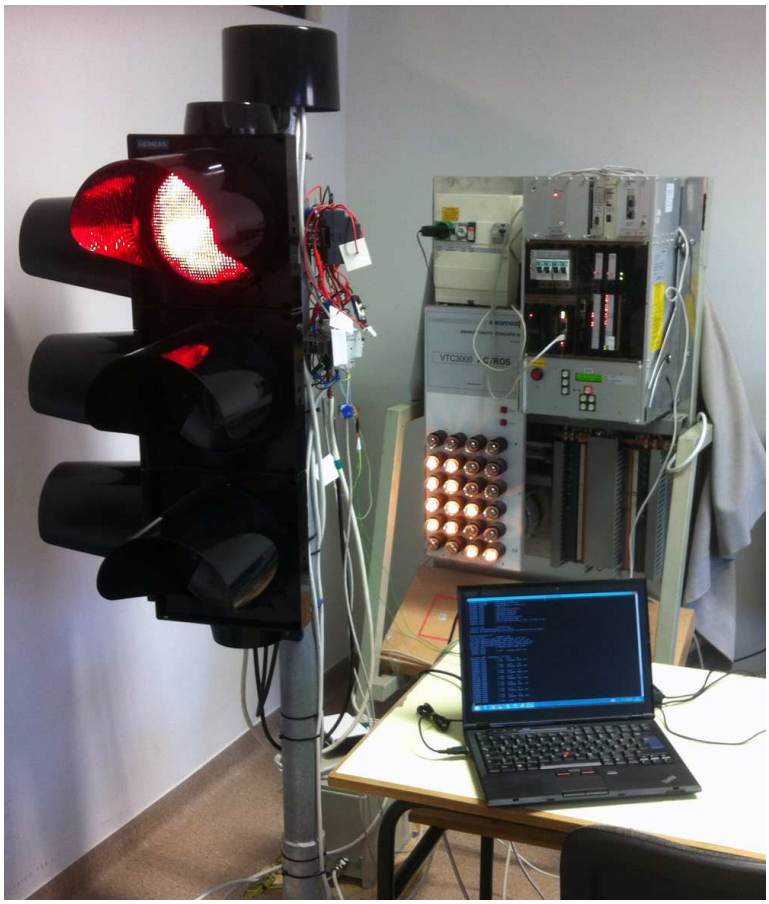
6.3. A rendszer működése

Ebben a fejezetben a rendszer működésének összefoglalója következik. A működéshez először minden eszközt feszültség alá kellett helyezni. Elsőként bekapcsoltam a labor routerét, hogy legyen hálózat, majd az 5.1 ábrán látható biztosítékokat felkapcsoltam. Ezzel elindultak a PLC-k, áram alá került az Elba jelzőfej, továbbá elindult a transzformátor, ami áram alá helyezte a Siemens jelzőfejet. A harmadik PLC (a világító diódás általam készített jelzőt irányító) elindításához a tápegységét bedugtam a hálózati táplálásba. A PLC-ket az Actrossal és hálózattal összekötő hub-ot is feszültség alá helyeztem. Majd elindítottam az Actrost is.
Hálózaton keresztül feltöltöttem rá a 'vt.jar' fájlt, ami az Eclipse-ből kiexportált 'src' mappa fájljait tartalmazta (lásd 6.5 ábra). Ezek után az Actrost újra kellett indítanom, hogy a friss fájllal induljon el. A kódok hibamentessége esetén az Actros el tudott indul és elkezdte kiküldeni a LOGO!-knak az utasításokat, melyek vezérelni kezdték a jelzők izzóit.
Az AltalanosResz.java, mivel a jelzési program része, ezért minden fél másodpercben lefut, és meghívja a FixProg1.java-t. A FixProg1.java-ban lévő kódsor az IP cím alapján azonosítva, PLC-nként sorban meghívja a LogoJelzofej.java BEKI függvényét, mely eldönti az időpillanat alapján, hogy melyik jelzésképet kell használni. Ezután szintén PLC-nként sorban, kiadja azt a set függvénynek, mely a main szálon kívül elindít egy új szálat és elküldi a LOGO!-nak az utasítást. Így fél másodperc alatt három új szál indul el, mert három logikai modulunk van. Ekkor fizikailag megtörténik a jelzőfej izzójának kapcsolása. Ezek a szálak a feladatuk elvégzésével leállnak és megszűnnek. A következő fél másodpercben ugyanez a műveletsor játszódik le. Eközben feszültség ellenőrzés céljából további feladato-

kat végeztetek a LogoJelzofej.java –ban. Erről a 7.3. fejezetben írok részletesen. A folyamatot a 6.6 ábrán követhetjük le. A megvalósított rendszer a 6.7 ábrán látható.



6.6. ábra. A működés folyamatábrája



6.7. ábra. A megvalósított rendszer

7. fejezet

A rendszer biztonsága

Napjainkban a biztonságot úgy definiáljuk, hogy egy olyan zavartalan állapot, amely veszélyektől, vagy bántódástól mentes [Juhász, 1972]. Közös cél, hogy a közúti közlekedésben is elérjünk egy ilyen biztonságos állapotot, melyhez elengedhetetlen, hogy a forgalomirányító berendezések hibátlanul végezzék a feladatukat. A projektemmel egy olyan új rendszert hoztam létre, mely egy meglévő átalakításával jött létre. Az átalakítás során a biztonsággal kapcsolatban meglévő feladatok ugyan megmaradtak, de megoldásukat újra kellett terveznem.
Egy forgalomirányító berendezés biztonsági funkciói a következők kell, hogy legyenek:

- a tiltott (ellentétes) jelzésképek felismerése, megjelenésük esetén a berendezés sárga-villogó üzemmódba való kapcsolása,
- belső hiba esetén hibabiztos állapotba kerüljön (hibától függően sárga-villogó vagy sötét üzem),
- zöld együtt égés tiltása – közbenső idők meglétének ellenőrzése,
- és a fénypontokon lévő izzók kiégésének ellenőrzése.

Ezen ellenőrzések és funkciók hiányában, súlyos balesetek következhetnek be, ezért meglétük elengedhetetlen. A hibák előfordulását nehéz kiküszöbölni. Míg például a közbenső időből eredő hiba tervezési probléma következménye lehet, addig egy izzó kiégés főleg műszaki probléma eredménye. Célom, hogy a rendszerem bármelyik eset bekövetkezését azonnal érzékelje, és az irányítást egy biztonságos állapotba vezérelje.
Belső hiba esetén a PLC-k sárga villogó funkcióba kapcsolnak, erről a 7.1. fejezetben írok. Zöld együtt égés tiltását a közbenső idők meglétével biztosítom. Ezek ellenőrzését a 7.2 fejezet mutatja be. Az izzók kiégése feszültség és áram ellenőrzéssel vizsgálható. Erről a 7.3 fejezetben lesz szó.

Ezek előtt elsőként bemutatom, hogy milyen módon működik a rendszeremben a sárga villogó funkció, majd az egyes hibaforrások felismerésére és kezelésére kialakított metódusokat ismertetem.

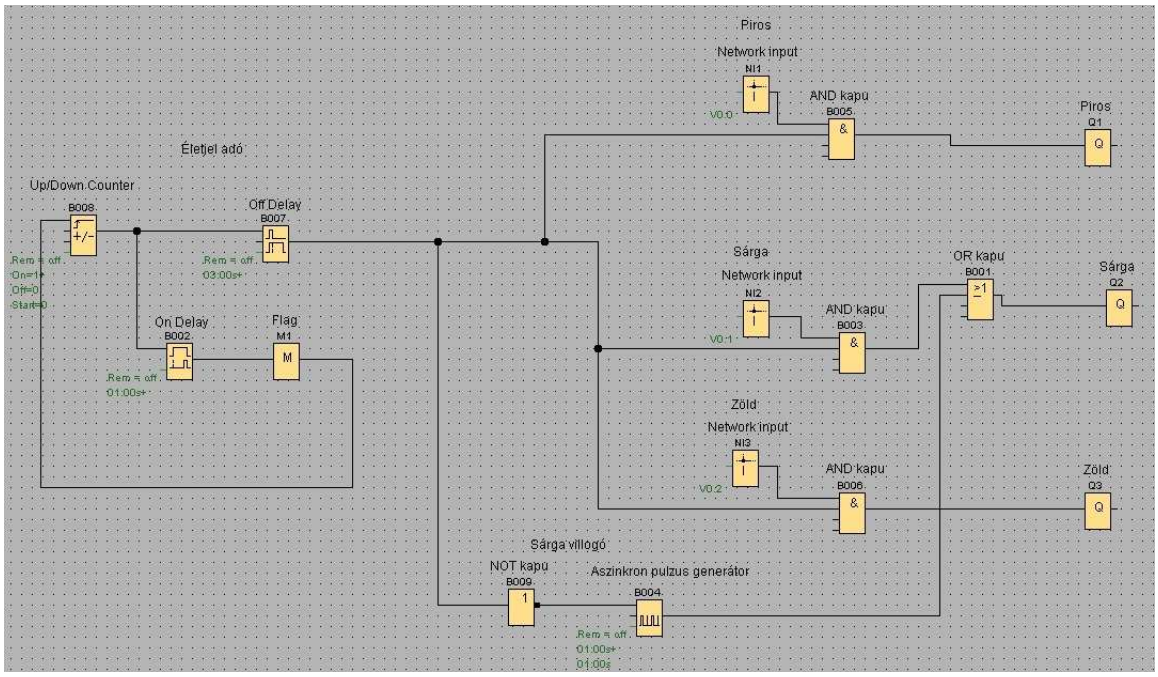
7.1. Sárga villogó üzemmód

A közúti közlekedés forgalomirányításában, a sárgán villogó jelző a rendszer üzemen kívüli állapotát jelenti. Ez lehet a meghibásodás, de olykor a jelzőlámpás irányítás szükségtelensége (például éjszaka lekapcsolt lámpák) miatt is. Ilyen esetben a jelzőfejek mellett elhelyezett közúti jelzőtáblák lépnek érvénybe, melyek általában a stop tábla és az elsőbbség adást kötelező tábla. Egy rendszerben nagyon fontos, hogy a felhasználó mindig értesüljön a hibáról. A sárga szín villogása figyelemfelkeltő. Továbbá teljesen elszeparálható a többi jelzésképtől, így a közlekedésben résztvevő azonnal tudja, hogy kereszteződésben fokozott figyelemre van szükség. A jelzőket sárga villogó üzemmódba (az éjszakai szándékos sárga-villogó üzemet kivéve) zavar esetén kell vezérelni, ami fizikai oldalról lehet például izzó kiégés, vezeték szakadás és bármilyen irányítási hiba fellépése is, mint például zöld együtt égés, vagy a forgalomirányító berendezés belső hibája.
Az intelligens jelzőfejjel működő rendszerben az egyik legfontosabb kapcsolat a LOGO!-k és az Actros között áll fenn. Ha ez a kapcsolat megsérül, akkor a forgalomirányító berendezés elveszítheti az irányítást a jelzőfejek fölött, így ennek biztonsága kulcsfontosságú. Ezért is választottam egy olyan megoldást, ami teljes mértékben garantálja, a kapcsolat meghibásodásával is biztonságos a kimenetet. Sárga villogó állapotot idézek elő, ha a PLC és az Actros között nincs kapcsolat, így nem történhet meg olyan, hogy egy kiadott jelzéskép a jelzőkön marad, mert megszakadt a kapcsolat és a következő utasítás már nem ér oda. Ha egy jelzéskép ki van adva, de valami miatt a kapcsolat megszakad, akkor a LOGO! azonnal sárga villogó üzembe vezérel. Ezt úgy tudtam megvalósítani, hogy a LOGO!-ra feltöltött programon módosítottam, azaz lényegében az Actros ki van hagyva ebből a biztonsági funkcióból.

7.1.1. A sárga villogó program tervezése Logo Soft Comfortban

A sárga villogó funkció kialakítása a következő módon történt. Az Actros és a PLC közötti kapcsolat meglétét vizsgáltam. Ha a kapcsolat fenn áll, akkor az Actros által kiadott jelzésképek futhatnak. Ha nincs kapcsolat, akkor a PLC átveszi az uralmat, és a rajta futó program sárga villogóra állítja a jelzőfejeket. A kapcsolat meglétét folyamatosan vizsgálnom kellett. Mivel az Actros fél másodpercenként lefuttatja a kódjait, ezért ezt ki tudtam használni. Egy jelet vizsgáltam, amit az

Actros küldött ki (ezt én generáltam), hogy megérkezik-e. Ennek menete az, hogy az Actrosban fél másodpercenként lefutó 'FixProg1' program mindig kiküldte ezt a vezérlőjelet a PLC felé, ami ezt mindig lenullázta. A kapcsolatot vizsgáló programban van egy ellenőrző visszacsatolás, az Actros által küldött jel nullázásának utolsó idejéről. Ha ez meghaladt egy bizonyos határértéket, amit én határoztam meg a biztonság függvényében - azaz például 2 másodperc óta nem érkezett jel az Actros felől - akkor átkapcsolt minden PLC-t sárga-villogó üzemmódba. Mindehhez át kellett alakítani az eddig meglévő LOGO!-ra feltöltött programomat, amit a 6.1. fejezetben ismertettem. Az újratervezett program blokk diagramja a 7.1 ábrán látható.



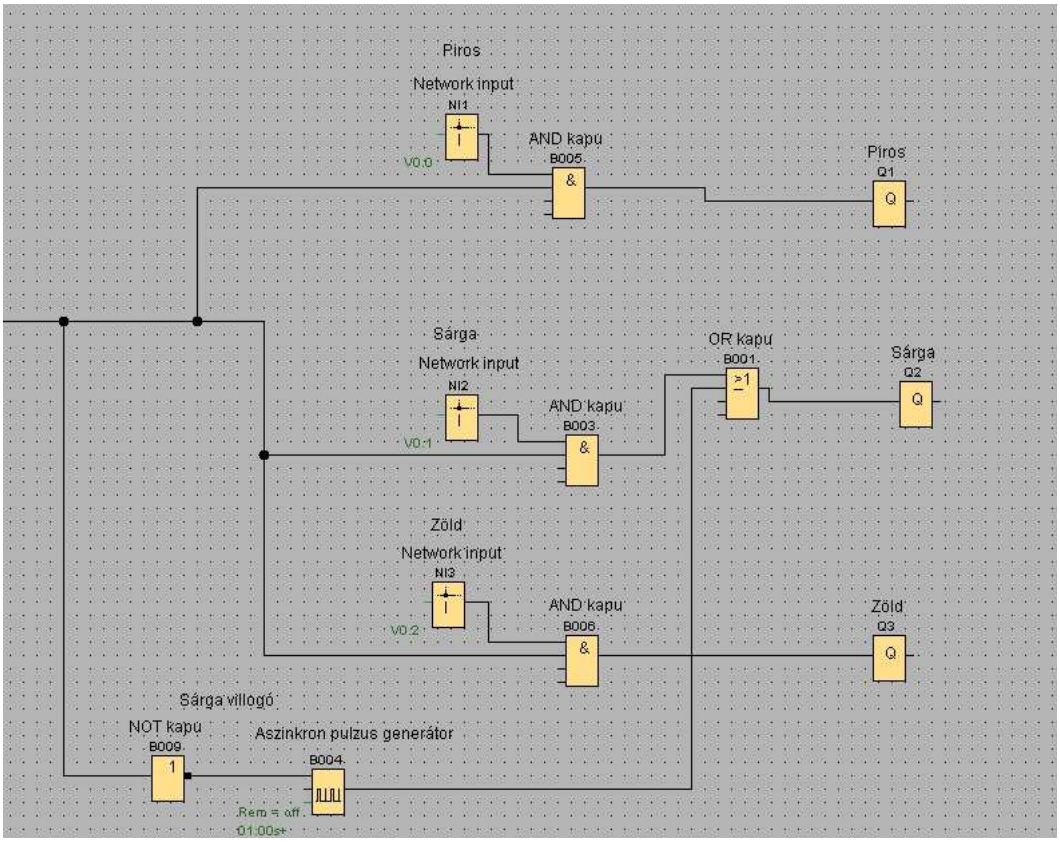
7.1. ábra. A sárga villogó funkcióval ellátott blokkdiagram

A következőkben a program egyes részleteinek magyarázata következik, a kimenetek oldaláról kezdve. Elsőként a 7.2 ábrán látható részletről írok.

- Ha érkezett ellenőrző vezérlőjel az Actros felől ÉS kiadtam a piros jelzést (Network Inputon keresztül az Actrosból), akkor kiengedem a jelet a Q1 kimenetre (ami fizikailag a piros izzónak felel meg).
- Ha érkezett a jel az Actros felől ÉS kiadtam a sárga jelzést (Network Inputon keresztül az Actrosból) VAGY, ha sárga villogó utasítás érkezett, akkor továbbengedem a jeleket a Q2 kimenetre (ami fizikailag a sárga izzónak felel meg).

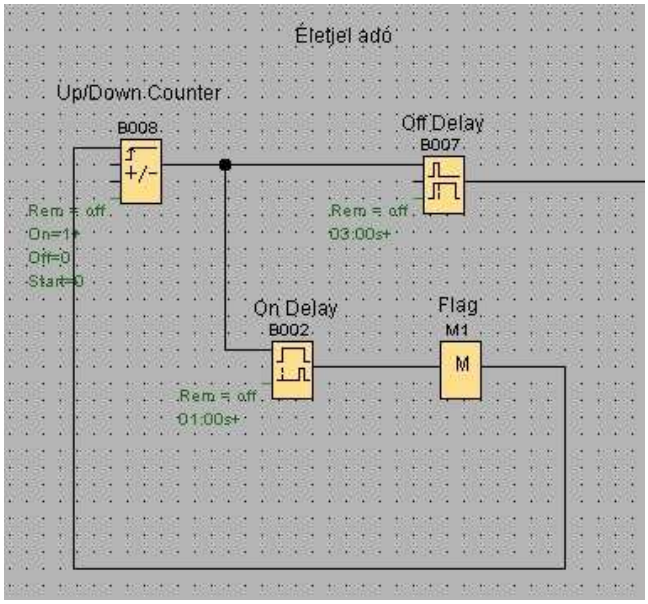
- Ha érkezett jel az Actros felől ÉS kiadtam a zöld jelzést (Network Input-on keresztül az Actrosból), akkor kiengedem a jelet a Q3 kimenetre (ami fizikailag a zöld izzónak felel meg).

A sárga villogó funkciónál, magáért a villogásért egy aszinkron pulzus generátor (Asynchronous Pulse Generator néven találjuk meg a Logo Soft Comfortban) felelt. Ezt a 7.2 ábra alján láthatjuk. A pulzus generátor a tulajdonságai között meghatározott időközönként megszakítja a jelünket, azaz pulzálást idéz elő. A blokkon való jobb egér gombbal kattintással, elérhető a Block Propertiest, ahol be lehet állítani az impulzus szélességét. Ezt 1 másodpercre állítottam. A 7.1 ábra



7.2. ábra. Részlet a blokkdiagramból I.

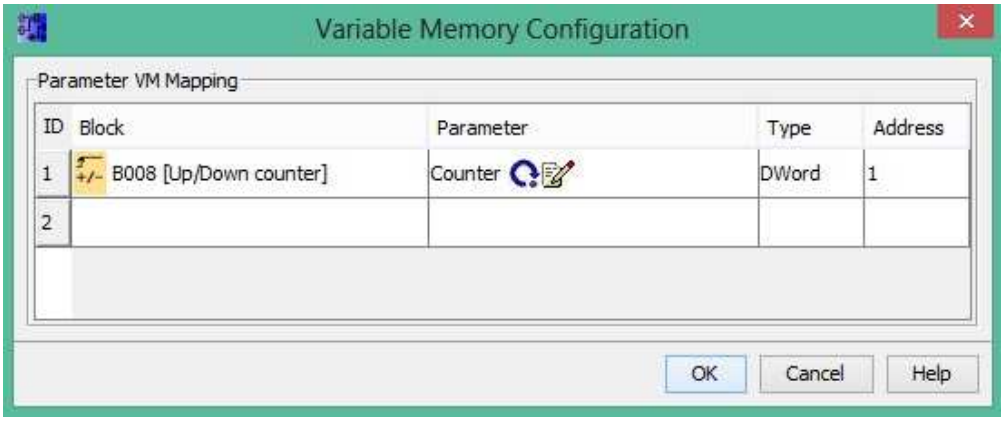
bal oldalán elhelyezkedő rész felel az Actros felőli kommunikációért. Ezt kinagyítva a 7.3 ábrán láthatjuk. Életjelet kellett generálnom az Actrosból, majd azt a PLC-n futó programmal ellenőriznem. Ehhez a Logo Soft Comfort VM Mapping funkcióját használtam ki. A magyarázatom értelmezéséhez először a Parameter VM Mapping-ról írok röviden.



7.3. ábra. Részlet a blokkdiagramból II.

Parameter VM Mapping

A VM (Variable Memory - lásd 1)egyes byte-jaihoz a PLC –n futó program elemeinek tulajdonságait lehet kötni. Ezáltal ezeket a helyeket a VM-ben a PLC minden frissítési ciklusban feltölti az aktuális értékekkel. A Logo Soft Comfortban a Tools menü alatt található „Parameter VM Mapping" ablakban lehet beállítani, hogy melyik blokk melyik tulajdonságát a VM-en belül hova írja ciklusonként a LOGO! (lásd 7.4). [Ludvig, 2013] Én az Up/Down Counter (Fel/Le Számláló) blokk VM-én állítottam.



7.4. ábra. A VM konfigurációja

A Block és a Parameter oszlopokban lehet kiválasztani a kívánt függvényblokkot és annak kívánt paraméterét. A típusát a Type oszlopban lehet módosítani,az Address fül alatt pedig beállítható a VM-en belüli címe [Ludvig, 2013]. A megvalósítás során Dword típust alkalmaztam - mert a program csak ezt kínálja fel - mely 4 byte-ot foglal el, a címének pedig 1-et adtam meg. Ez lényegében bármi lehet, csak figyelni kell, hogy az Eclipse-ből majd a beállított címre kell küldeni a vezérlőjelet.

Térjünk vissza a 7.3 ábrára. Az Up/Down Counteren keresztül érkezik a jel az Actros felől, amit az Eclipse-ben adtam ki utasításként. Mögötte elhelyeztem egy Off Delayt, ami úgy működik, hogy ha ráengedem a jelet, az átmegy rajta. Ha megszűnik a jel az inputján, az outputon még marad jel (lényegében húzva marad), és elindul benne egy visszaszámláló (aminek az értékét a blokk tulajdonságaiban lehet beállítani). Ha lejár a visszaszámláló, akkor megszűnik a jel az outputon is. Az Off Delay-re 3 másodperces visszaszámlálást állítottam be. Így az Actrosból fél másodpercenként érkező jel folyton elindítja, de visszaszámlálás csak fél másodpercig jut el a háromból, mert érkezik a következő jel (feltéve, ha a kapcsolat még fenn áll, de pont ezt vizsgálom).
A Counter mögé közvetlenül beraktam egy visszacsatolást, melyre egy On Delayt kötöttem. Ez a visszacsatolás fogja belökni az Off Delayt a sárga villogó funkcióba, hogyha nem érkezik több jel az Actros felől. Az On Delay úgy működik, hogy mikor rámegy a jel, akkor elindul benne egy visszaszámláló. Ha letelik a beállított idő, akkor a blokk tovább engedi a jelet. 2 másodpercre állítottam az On Delay blokk visszaszámlálási idejét. Így a fél másodpercenként érkező Actros jel mindig újraindítja benne a visszaszámlálást. Amíg az Actrossal van kapcsolat, addig a visszacsatoláson nem megy jel, mert az On Delay nem engedi át. Ha megszűnik a jel az Actros felől és letelik az On Delay-ben a visszaszámlálás, akkor továbbengedi az utolsó hozzá érkezett jelet, ami eljut Up/Down Counter-re és lekapcsolja. A megszűnő jel lekapcsolja az Off Delayt is, és elindítja a sárga villogó funkciót.
A VM írására az alábbi függvényekkel van lehetőség:

- Logo setByte(intpos, intvalue)
- Logo setWord(intpos, intvalue)
- Logo setDWord(intpos, intvalue) [Ludvig, 2013]

Lehet byte-ot, szót (Word) és dupla szót (Dword) írni a VM-be. Én az alábbi paranccsal írtam a VM-re:

```
logo_i.setDWord(1, 10);
```

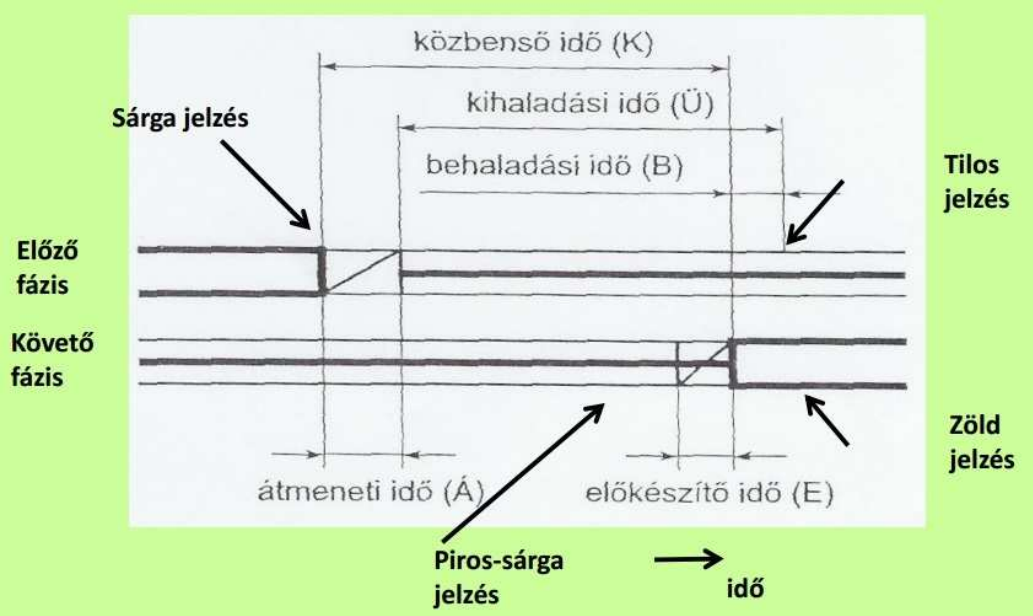
Itt látható, hogy a Logo Softban a VM Configuration-ban beállított 1. címre írtam 10-es értéket. Az Eclipse-ben egy olyan helyre kellett raknom a vezérjelet generáló kódot, ami minden fél másodpercben lefut. A BEKI függvényem ilyen, mely a LogoJelzofej.java-ban van megírva. A BEKI-ben indítottam egy új szálat a main szál és a set függvény-ben indított szál mellé. Ez a szál minden fél másodpercben elküldi a setDWord(1, 10); paranccsal a vezérlőjelet a LOGO!-knak.

7.2. Közbenső idők meglétének ellenőrzése

A csomóponton áthaladó forgalmi áramlatok forgalombiztonsági okból közvetlenül nem követhetik egymást. Az egymást keresztező, vagy nyomvonal szempontjából kapcsolódó (fonódó) mozgások szabad jelzései között biztosítandó legkisebb időt közbenső időnek nevezzük. A közbenső idő három tényezőből áll, mely:

- átmeneti – sárga jelzés – idő
- kihaladási – ürítési – idő
- behaladási idő [Debreczeni, 2013]

Értéke úgy számítható, hogy az átmeneti időhöz hozzáadjuk a kihaladási időt, majd kivonjuk a behaladási időt (lásd 7.5 ábra).
Az intelligens jelzőfejjel működő rendszer minden fél másodpercben ellenőrzöm a közbenső idők meglétét. Ha ebben hiba van, akkor a rendszert sárga villogó üzemmódba vezérlem. A LogoJelzofej.java-ban írtam egy ellenőrző részt, amely a BEKI függvényünkben megadott belépési és kilépési időből számítja a közbenső időt. Ezeket manuálisan számítottam, így ha további LOGO! kapcsolnék a rendszerbe, ezt a programrészt ki kellene egészíteni. A rendszeremben a kilépési és a belépési idő különbségeként meghatározható a közbenső idő (ezeket az időket függvénnyel kértem le). Ez nem minden esetben eredményezett pozitív számot, ezért vettem a művelet utáni érték abszolút értékét.



7.5. ábra. A közbenső idő részei [Debreczeni, 2013]

Ezt minden variációra elvégeztem és külön változókba mentettem őket. A kód itt látható, 3 PLC esetén 6 variáció, 6 változó:

```
kozb01 = Math.abs((Var.logo.getKezdes()-Var.logo1.getBefejezes()));
kozb02 = Math.abs((Var.logo.getKezdes()-Var.logo2.getBefejezes()));
kozb10 = Math.abs((Var.logo1.getKezdes()-Var.logo.getBefejezes()));
kozb12 = Math.abs((Var.logo1.getKezdes()-Var.logo2.getBefejezes()));
kozb20 = Math.abs((Var.logo2.getKezdes()-Var.logo.getBefejezes()));
kozb21 = Math.abs((Var.logo2.getKezdes()-Var.logo1.getBefejezes()));
```

Ezek után minden közbenső idő megvan egy változóban, melyeket az előre megadott (amit a csomópont tervezője által előírt) közbenső idő értékekhez hasonlítottam relációval. Ha ezek közül egy is rossz volt, azaz kisebb az általam számított közbenső idő, mint a megadott, akkor a rendszert sárga villog üzemmódba kapcsoltam. Ennek metódusa során az Actrosnak a LOGO!-kkal való kapcsolat bontása utasítást adtam. Az ellenőrzésre használt kódsor:

```
 if(kozb01>3 & kozb02>3 & kozb10>3 & kozb12>3 & kozb20>3 & kozb21>3){
       System.out.println("A kozbenso idok rendben vannak");}
else{
       System.out.println("A kozbenso ido valahol NEM OK - ezert sarga
                          villogo!");
       logo1.closeConnection();}
```

A kapcsolat bontása sárga villogó üzemmódot eredményez, mert nem küld az Actros vezérlőjelet a PLC-nek. Ezt már a a 7.1. fejezetben részleteztem.

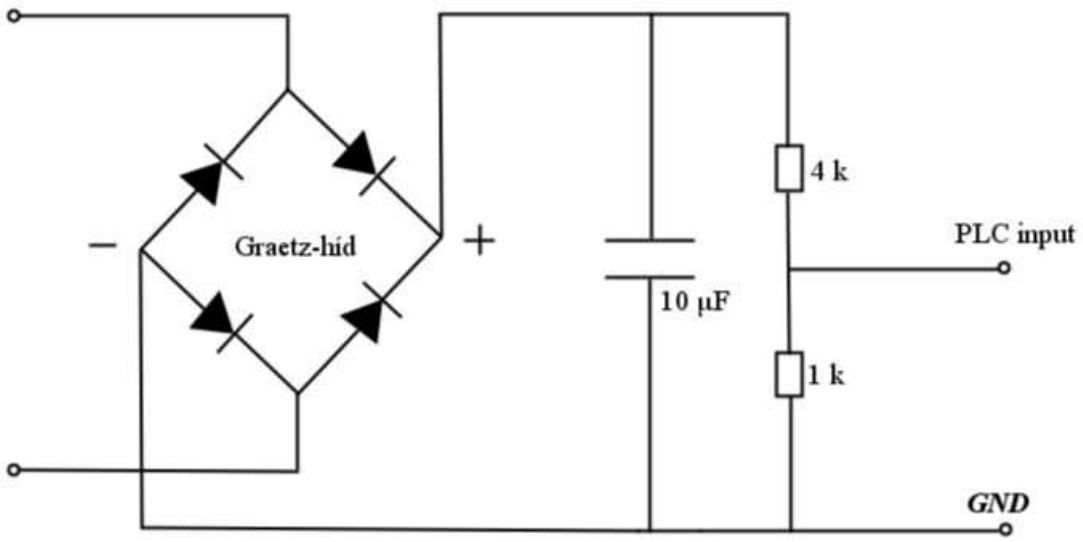
7.3. Feszültség ellenőrzése

Az izzók kiégésének ellenőrzését újra kellett tervezni, hiszen a forgalomirányító berendezéshez már nem kapcsolódnak az izzók elektromos kábelei. Az Actros a jelzőfejekkel csak egy UTP kábellel áll kapcsolatban, amin információkat továbbít, és fogad. A zöld izzók égését feszültség meglétével kell vizsgálni. A LOGO! 12/24 RCE a bemenetein (inputok) tud feszültséget ellenőrizni, mégpedig úgy, hogy az inputon lévő feszültséget a földponthoz hasonlítja. A specifikáció szerint, ha az inputon megjelenő feszültség kisebb, mint 5 V, akkor a válasza '0' lesz, ha az inputon megjelenő feszültség minimum 8,5 V, akkor a válasza '1' lesz. Így a PLC-től kapott válasz boolean típusú, azaz a válasz 'true' ha a zöld izzó ég és 'false' ha a zöld izzó nem ég.
A feszültség lekérdezését a programomban a BEKI függvénybe írtam bele. Több függvény is rendelkezésre áll a Logo olvasásához, én a 'boolean getInput (int i)' függvényt használtam. Az inputok sorszámozása 0-tól indul és mivel a zöld izzót akartam vizsgálni, ezért a 2. inputot kellett lekérdeznem, illetve ide kellett visszakötnöm a zöld izzót. Ahhoz, hogy az ellenőrzést nyomon tudjam követni kiírattam a parancsot a képernyőre a következő kóddal, mely magába foglalja a PLC bemenetének lekérdezését is:

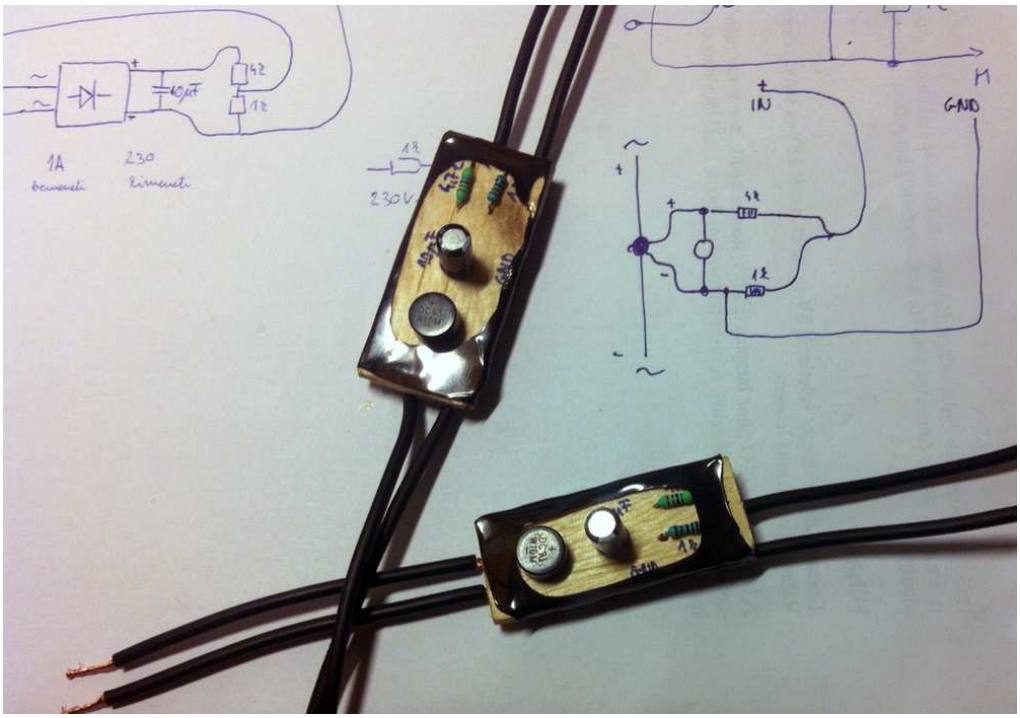
```
System.out.println(logoszama + " PLC " + "zoldeges " +
    logo_i.getInput(2)+" " + Var.tk1.getProgZeit());
```

A 'logoszama' változó megmondja, hogy melyik PLC-ről való ellenőrzését írattam ki. Ezt a lekérdezést szintén az előbb BEKI-ben elindított újabb szálba tettem bele. Erre azért van szükség, mert itt is a LOGO!-tól kértem egy választ, ami a 3 PLC esetén nem férne bele a fél másodperces futási időbe.

A jelzőfejek zöld izzóinak visszakötésénél ügyelnem kellett arra, hogy az izzókba 40 V-os illetve 230 V-os váltakozó áram megy ki. Ha ezt visszakötöm a PLC-be a váltakozó áramot fogja mérni. Így előfordulhat, hogy feszültség esetén is hamis választ kapunk, mert a szinusz görbe egy rossz pontjára mért rá a PLC. Emiatt egyenirányítanom kellett a váltakozó feszültséget. Ezt egy Graetz-híd és hozzá kapcsolódó kondenzátor és ellenállások segítségével tettem meg. A kapcsolási rajzot és a 7.6 ábra szemlélteti, melyen az ellenállások értéke a 40V-os jelzőfejhez készített egyenirányító értékeit mutatják. A 230V-os jelzőhöz készített egységnél két darab 1 kiloohmos ellenállást használtam fel. A megvalósított egyenirányítók a 7.7 ábrán láthatóak.



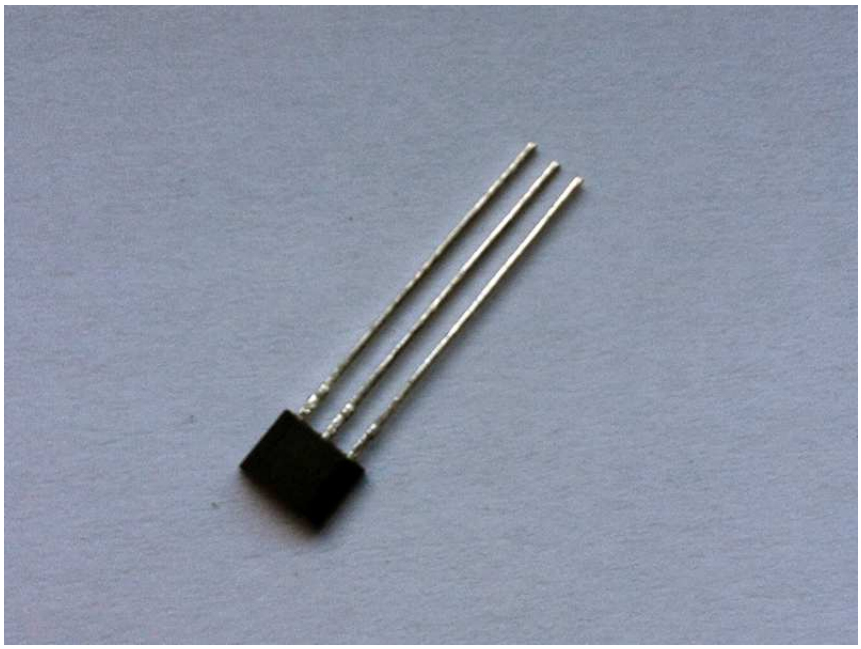
7.6. ábra. Az egyenirányító kapcsolási rajza



7.7. ábra. A megvalósított egyenirányítók

7.4. Áram ellenőrzés

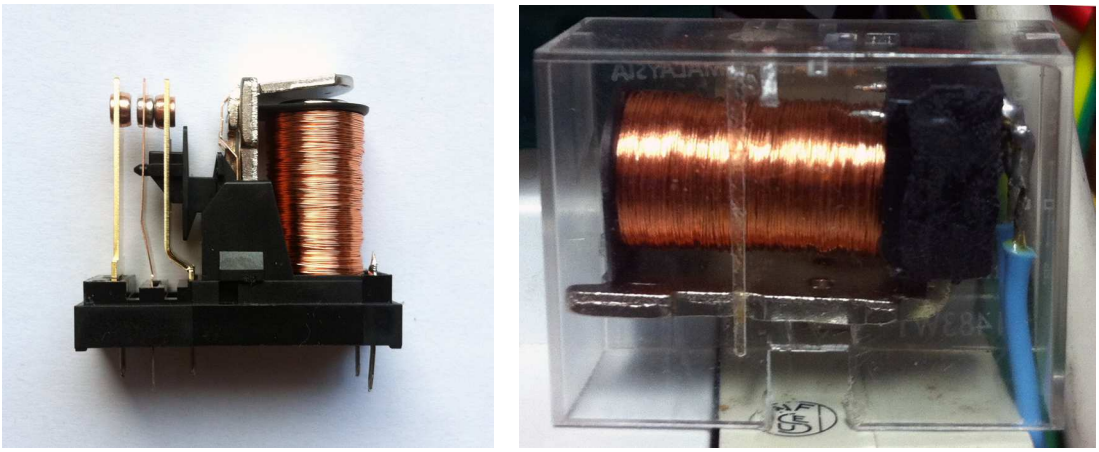
A jelzőlámpák piros izzóinak működését, illetve kiégését áram ellenőrzésével szokták vizsgálni. Rendszeremben a Hall-effektust használtam fel áram mérésére. A Hall-effektus az Edwin Hall által 1879-ben felfedezett jelenség, mely szerint, ha egy vezetőben vagy félvezetőben áram folyik, és azt mágneses térbe helyezzük, akkor az áramot hordozó részecskékre (fémeknél elektron) Lorentz-erő hat, ami azzal jár, hogy a vezető két oldalán feszültségkülönbség keletkezik. Ezt a feszültséget Hall-feszültségnek nevezik.
Kiskereskedelmi forgalomban Hall-szenzorok széles választékban megvásárolhatóak. Én egy Honeywell SS495A típusú szenzort (lásd 7.8) használtam fel a mérésemnél, mely egy nagyon kis méretű, alacsony fogyasztású, DC táplálást igénylő, 3 lábú (két láb a tápláláshoz, egy pedig Hall-feszültség méréséhez), analóg kimenetű egység. Az analóg kimenet fontos paraméter volt a jelfeldolgozás miatt. A LOGO! 12/24 RCE 4 analóg bemenettel rendelkezik, így ennek olvasására alkalmas. Az áram mérés funkciót csak a 40V-ról működő jelzőfejre valósítottam meg.



7.8. ábra. Hall-szenzor

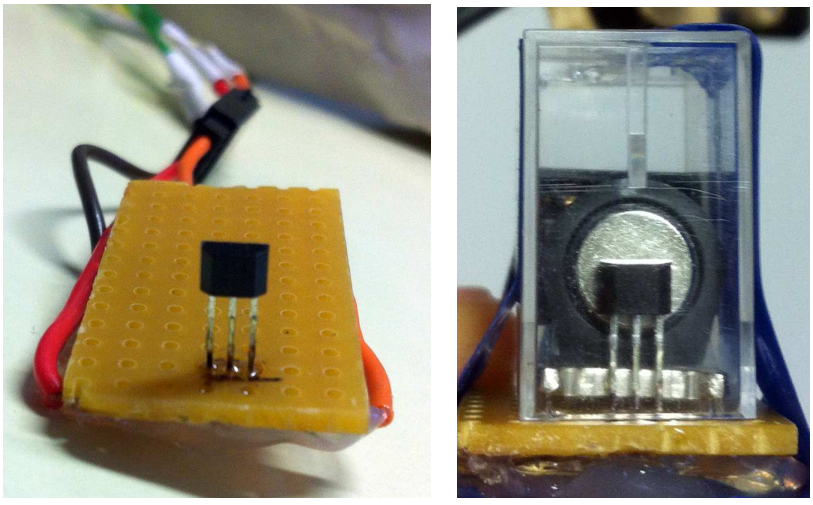
A Hall-szenzorral való méréshez először mágneses teret kellett gerjeszteni. A piros izzó előtti vezetéken 150 mA áram folyik az izzó működésekor. Ezt az erősséget kell a Hall-szenzoros árammérővel kimérni. Adott árammal vasmag alkalmazása mellett nagyobb mágneses indukciót lehet létrehozni, ezért a mágneses tér indukálására egy relét választottam, melynek csak az elektromágnes részét használtam

fel az árammérő megépítésekor. Egy 5V vezérlőfeszültségű, 110 mA áramerősségre záró jelfogót sikerült beszereznem, mely könnyen bontható, így az elektromágnes részét ki tudtam szerelni belőle (lásd 7.9).



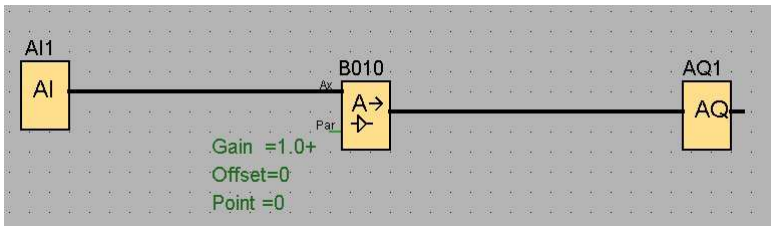
7.9. ábra. Az eredeti jelfogó, és a kiszerelt elektromágnes

Az árammérő egységet úgy kellett felépíteni, hogy az elektromágnes a Hall-szenzorra nézzen, ezért egy nyák lemezre építettem, és műanyag borítást kapott a szigetelés érdekében. A Hall-szenzort 5V DC árammal tápláltam, és az információt adó lábát egy egyenirányítón keresztül a PLC bemenetére kötöttem vissza. Az egyenirányítást itt is hasonlóan oldottam meg, mint ahogy azt a feszültség ellenőrzés esetén tettem (lásd 7.3 fejezet), csak feszültségosztó nélkül.



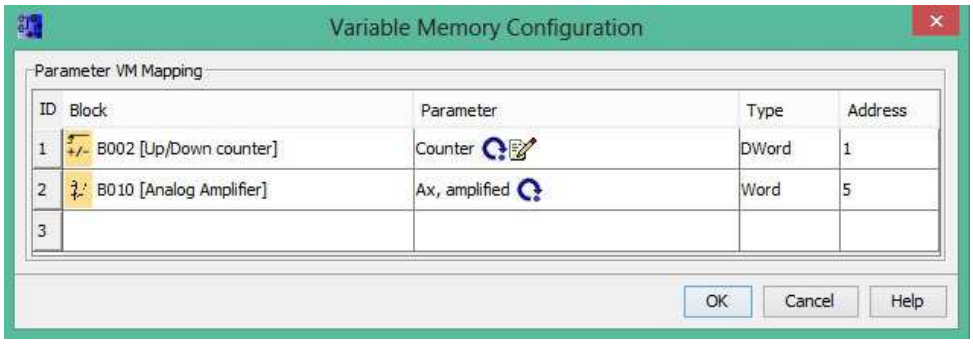
7.10. ábra. A Hall-szenzor, és az épített árammérő egység

Áram ellenőrzés esetén analóg jelet vezettem vissza a PLC bemenetére. A LOGO!-nak van AD átalakítója, így a jelet olvasni tudtam az Actroson keresztül. Az áram ellenőrzésekor, az elektromágnesre 150 mA folyik, mely mágneses mezőt gerjeszt maga előtt. Ezt a Hall-szenzor érzékelte és adott mágneses erősségre adott feszültségértéket ad. Így a PLC-re feszültséget vezetek vissza. Ezt a PLC a föld pontjához hasonlítja és ebből áll elő az eredmény.
Az analóg jel kiolvasásához egy blokkdiagramot kellett feltölteni a PLC-re a meglévő program mellé. Ennek elemei egy analóg bemenet, egy analóg erősítő és egy analóg kimenet. Erre azért volt szükség, mert a PLC bemenetére visszakötött analóg jelet csak Parameter VM Mapping-al tudtam kiolvasni, és ehhez kellett egy olyan blokk, ami erre megfelelő. Az analóg erősítő Parameter VM Mappingolásra megfelelő, ezért használtam fel. Erősítési funkciója nem szükséges a blokkdiagramomban, csak jelkiolvasására használom, így erősítését 1-re állítottam be. Az analóg jel olvasására kialakított kapcsolást a 7.11 ábra szemlélteti.



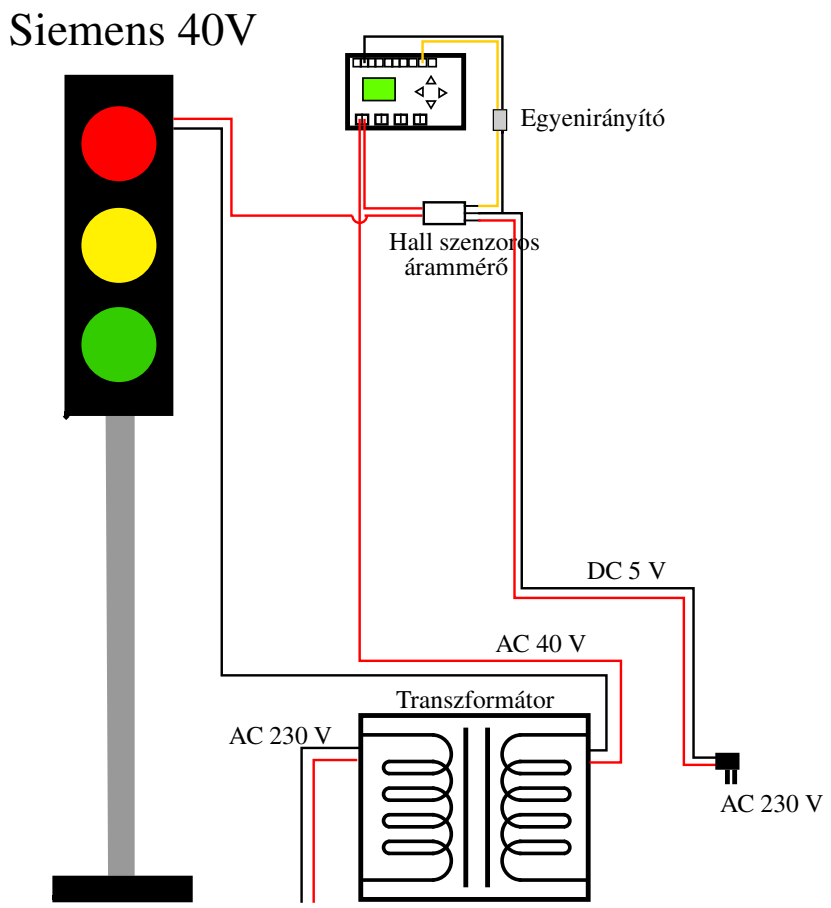
7.11. ábra. Analóg jel olvasására való blokkdiagram

A Logo Soft Comfortban be kellett állítanom a az analóg erősítő értékét. Ezt a tulajdonságaiban a Gain érték 1-re állításával tettem. Ezek után a Parameter VM Mappingban egy új sorban kiválasztottam az erősítő blokkot, majd a paraméterének beállítottam az "ax, amplified", azaz az erősíteni kívánt jelet. Típusa Word, Címe pedig 5 lett (lásd 7.12). Ezzel az analóg bemeneten érkező jelet le tudtam már kérdezni az Actroson keresztül.



7.12. ábra. Parameter VM Mapping beállítások

Továbbá fontos, hogy a Hall-szenzorról az egyenirányítón keresztül érkező jelet a PLC 7. bemenetére kötöttem vissza. A LOGO! 12/24 RCE 8 bemenetéből 4 analóg bemenetként is tud funkcionálni. Az első számú analóg bemenet a fizikailag 7. bemenettel egyezik meg. Valamint a blokkdiagramban az analóg bemenet blokkomat a tulajdonságai között AI 1-re állítottam. Az árammérő bekötési rajza a 7.13 ábrán látható.



7.13. ábra. Az árammérő bekötési rajza

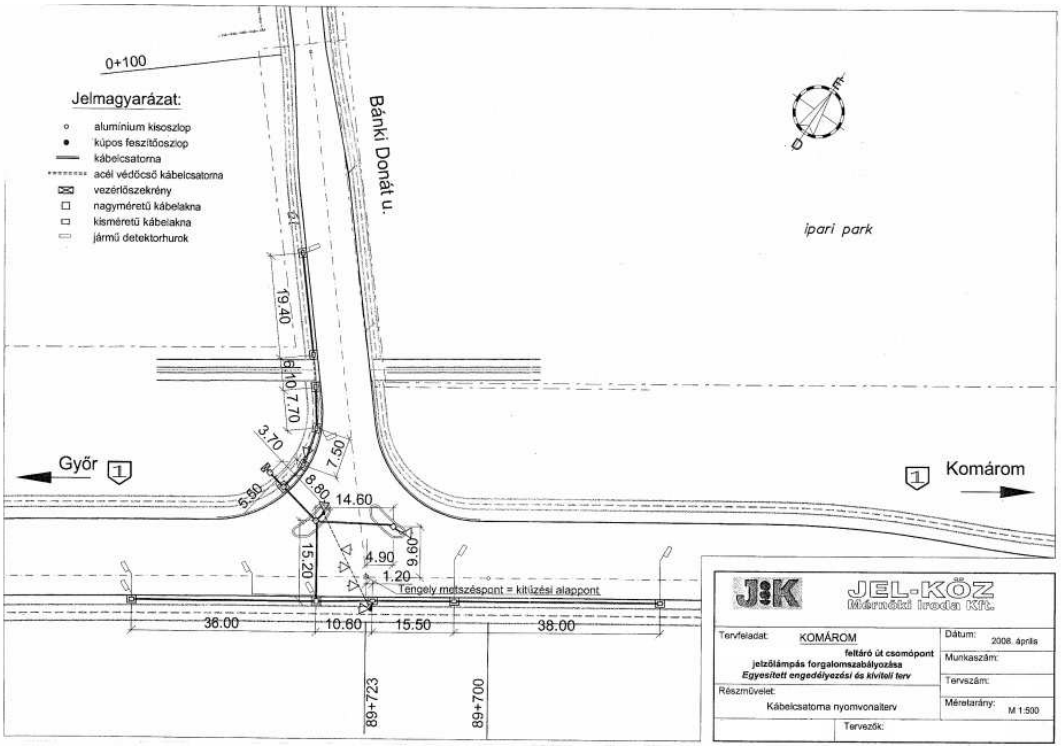
Az Eclipse-ben a LogoJelzofej.java -ban a BEKI függvényben a feszültség ellenőrzésre szolgáló kód utána írtam az áramellenőrzésre szolgáló részt. Az alábbi paranccsal tudtam lekérni a mért értéket:

```
System.out.println(logoszama + " PLC " + "piros " + logo_i.getWord(5)+"
    " + Var.tk1.getProgZeit());
```

8. fejezet

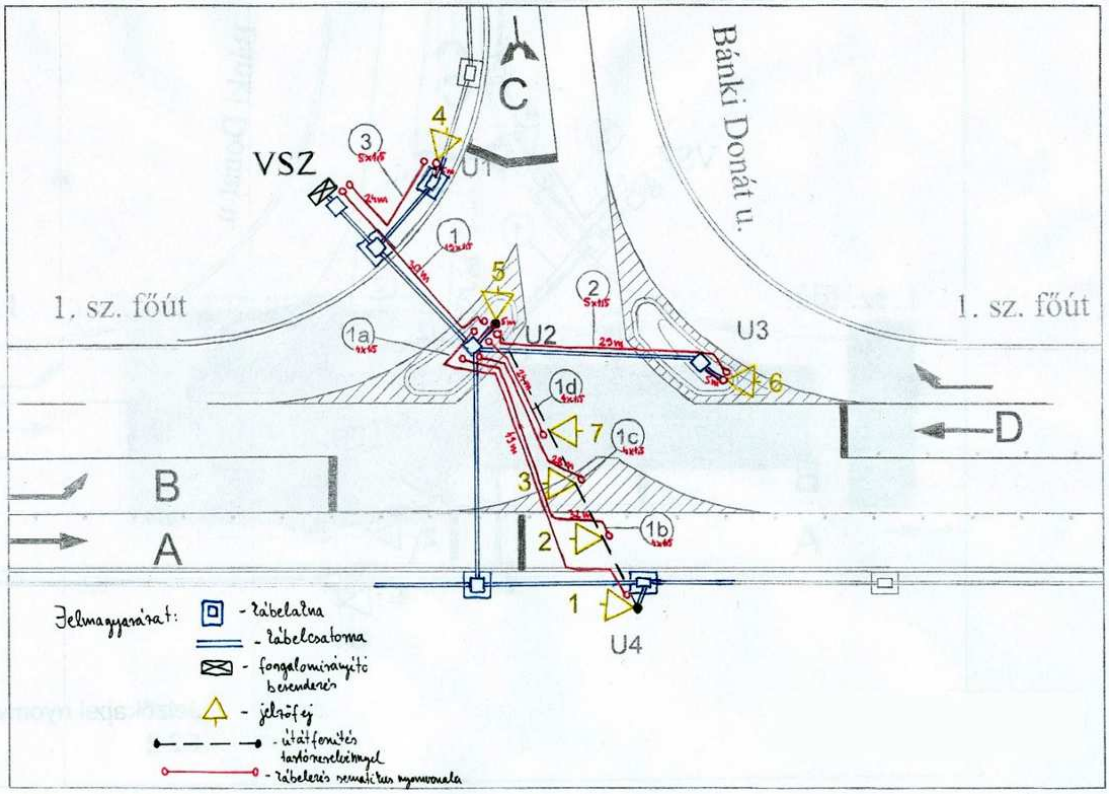
A fejlesztett rendszer értékelése

Egy új rendszer hasznosságának, illetve hatékonyságának megbizonyosodásához, össze kell hasonlítani egy már létező rendszerrel. Az intelligens jelzőfejjel működő rendszert az általános forgalomirányító rendszerrel tudtam legkönnyebben összevetni. A két különböző kialakítás előnyeit és hátrányait vizsgáltam meg. Az intelligens jelzőfej funkcionalitásban nem lépett előrébb a korábbi rendszernél, hiszen nem tud többet, mint az elődje. Ellenben gazdasági előnyei vannak. Legfőképpen a telepítési költsége, de emellett a károkból eredő javítási költsége is kevesebb.
Összehasonlításként egy T-alakú csomópont kialakításának költségét vizsgáltam meg a jelenlegi rendszerre, aztán az intelligens jelzőfejjel működő rendszer esetére. A csomópont egy valóságos kereszteződés, mely Komárom mellett található. A csomópont kialakítását és forgalomirányító eszköz szükségletét a Swarco Traffic Hungary Kft. tervezte még 2008-ban. Ennek teveit és költségvetését elkértem a cégtől, hogy egy olyan konkrét kereszteződést tudjak az összehasonlítás alapjává tenni, ahol általános forgalomirányítás zajlik. A csomópont a 8.1 ábrán látható.
Az építési költségek vizsgálatánál csak azokat a pontokat szedtem össze, amiben a két rendszer eltér. Összegyűjtöttem a tervek alapján a kiépítéshez szükséges kábelek típusait és mennyiségüket. Ebben a rendelkezésemre álló nyomvonalterv segített, melyet a 8.2 ábra szemléltet. A 8.2 ábrán látható egy VSZ felirat. Ez a forgalomirányító berendezés helye. Az elektromos művek ide ad tápellátást. Innen kell eljuttatni az áramot a jelzőfejekhez. A kábeleket nem lehet bárhol vezetni és bárhogyan. Erre kábelcsatornákat kell készíteni a föld alatt. Előfordulhat, hogy az utat többször is át kell vágni, hogy ezek a csatornák a megfelelő helyen húzódjanak. A csatornákba vonóvezetékkel húzzák be a kábeleket. A csatornák kábelaknákba vezetnek bele. Ezekben az aknákban végzik a kábelek behúzását, összekapcsolását, le- és szétágaztatását. Minden olyan helyen, ahol a kábelek elágaznak, egymásba futnak, egy-egy ilyen aknát kell építeni. A 8.2 ábrán ezeket az aknákat kék színű téglalap jelzi, a kábelcsatornákat meg szintén kék színű két párhuzamos vonal. Ahol kábelcsatornára nincs lehetőség, ott átfeszítést alkalmaznak. Átfeszítés kell



8.1. ábra. T alakú csomópont tervrajza [Bodó, 2008]

az út fölé belógatott jelzőfejeknek. Ehhez külön tartószerelvényt kell építeni. A kábeleket a sodronykötélen vezetik el a jelzőfejekig. Ebben az esetben különleges UV álló kábeleket alkalmaznak. A 8.2 ábrán az útátfeszítést fekete szaggatott vonal jelzi. Az 1. sz. főút fölé kellett jelzőfejeket belógatni. A 8.2 ábrán számmal ellátott sárga háromszög jelzi a jelzőfejeket. A kábelezést piros vonal jelöli, a vonalon lévő szám a hosszúságukat mutatja. Ezek a vonalak nem ott helyezkednek el, ahol a valóságban futnak a kábelek – csatornában vannak -, de a jobb átláthatóság érdekében ezeket máshol ábrázolták.



8.2. ábra. Jelzőkábelek nyomvonalrajza [Bodó, 2008]

A Swarco-tól kapott műszaki dokumentáció tartalmazza a kábelek pontos típusát és hosszát. Ez a 8.3 táblázatban látható. Továbbá kaptam a cégtől egy költségvetés kivonatot is, mely tartalmazza a kábelek árait. Ezek itt láthatóak:

- SZRMKVM – J 4x1.5 mm2 – 323 Ft (nettó ár)
- SZRMKVM – J 5x1.5 mm2 – 385 Ft (nettó ár)
- SZRMKVM – J 19x1.5 mm2 – 1056 Ft (nettó ár)

A J 4x1.5 mm2 például azt jelenti, hogy ez egy 4 erű kábel. A csomópontban 4, 5 és 19 erű kábeleket használtak fel. Nagyon sok erű kábelt általában akkor használnak, ha egy távolabbi pontra viszik a kábeleket. A VSZ és az U2 pont között vezet 19 erű kábel, mert az U2-től sok felé kell szétágaztatni. (lásd 8.2) A kábelezés költsége ezek után kiszámolható.

C = 148 m $*$ 323 Ft + 53 m $*$ 385 Ft + 30 m $*$ 1056 Ft = 47804 Ft + 20405 Ft + 31680 Ft = 99889 Ft (nettó ár)
Kijelenthető, hogy az általános forgalomirányítás kábelezésének költsége ennél a T-alakú csomópontnál körülbelül nettó százezer forint.

Honnan	Hová	Kábel szám	Kábel típus		Hossz	
VSZ	U2 oszlop	1	SZRMKVM-J	19 x1,5 mm2	30	m
U2 oszlop	1 fej (átfeszítésen át)	1a	SZRMKVM-J	4x1.5 mm2	49	m
U2 oszlop	2 fej	1b	SZRMKVM-J	4x1.5 mm2	32	m
U2 oszlop	3 fej	1c	SZRMKVM-J	4x1.5 mm2	28	m
U2 oszlop	7 fej	1d	SZRMKVM-J	4x1.5 mm2	24	m
U2 oszlop	5 fej	1e	SZRMKVM-J	4x1.5 mm2	5	m
U2 oszlop	U3 oszlop	2	SZRMKVM-J	5x1.5 mm2	29	m
U3 oszlop	6 fej	2a	SZRMKVM-J	4x1.5 mm2	5	m
VSZ	U1 oszlop	3	SZRMKVM-J	5x1.5 mm2	24	m
U1 oszlop	4 fej	3a	SZRMKVM-J	4x1.5 mm2	5	m
	földháló		alu sodrony	50 mm2	90	m
			Összesítés:			
			SZRMKVM-J	4x1.5 mm2	148	m
			SZRMKVM-J	5x1.5 mm2	53	m
			SZRMKVM-J	19x1.5 mm2	30	m
			Alu. Sodrony	50 mm2	90	m

8.3. ábra. Kábelek típusai és mennyiségük [Bodó, 2008]

Ezek után kiszámoltam, hogy mennyi lenne ennek a csomópontnak kábelezési költsége, ha intelligens jelzőfejekkel építenék ki a csomópont irányítását. A kábelek legnagyobb részétől meg lehet válni. A jelzőfejek között ki kell alakítani egy gerinchálózatot egy SZRMKVM-J 4x1.5 mm2 –es kábellel. Ez látja el a logikai modulok tápegységeit árammal. Ezen kívül már csak UTP kábelre van szükség a jelzőfejek és a forgalomirányító berendezés közé. Az intelligens jelzőfejjel irányított rendszer kábelszükségletei a 8.1 táblázat mutatja sorban az egyes pontok között véve (értelmezésben 8.2 ábra segít):

Honnan	**Hova**	**Kábel típus**	**Hossz** [m]
VSZ	U2	UTP	30
U2	U4	UTP	36
U2	U3	UTP	29
U3	6. jelzőfej	UTP	5
U2	5. jelzőfej	UTP	5
VSZ	U1	UTP	14
U1	4. jelzőfej	UTP	5
Összesen:			**124**

8.1. táblázat. UTP kábel szükséglet

Az U2 –től U4-ig való vezetés magába foglalja a 7., 3., 2., és 1. jelzőfej érintését is! A logikai modulok tápellátására bőven elegendő a 4 erű kábel. Ennek szükséglete körülbelül megegyezik az UTP kábelek hosszával. Az UTP kábel átlagára 150 Ft/méter. Intelligens jelzőfej esetén a kábelezés költsége így:

C = 124 m ∗ 150 Ft + 124 m ∗ 323 Ft = 18600 Ft + 40052 Ft = 58652 Ft (nettó ár)

Ebből a számításból látható, hogy az intelligens jelzőfejjel működő rendszer nettó kábelezési költsége valamivel több, mint ötvenezer forint. Ez 58,71 %-a az általános rendszer kábelezési költségének. Ez igen meggyőző lehet egy befektető számára a napjaink gazdasági helyzete mellett. Természetesen a PLC-k árát sem felejtethetjük el, és ez nincs benne ebben a számításban, mégpedig a következők miatt.

Az intelligens jelzőfejjel működő rendszer megalkotásának kezdetekor több variáció is volt arra, hogy milyen logikai modult használjak majd fel a projektben. A tanszéki labor két lehetőséget kínált ebből az egyik volt a Siemens PLC-je, a másik pedig egy BeagleBoard[1]. A relékimenetek könnyű programozhatósága miatt a PLC-t választottam, és a rendszert sikerült működőképesre megalkotnom. Ezzel igazoltam, hogy az intelligens jelzőfej további korszerűbb és akár költséghatékonyabb rendszert nyújt a közúti forgalomirányításban. Az általam felhasznált LOGO! 12/24 RCE nem tartozik az olcsó PLC modulok közé. Tudásának csak egy nagyon kis részét használtam ki. Egy hasonló paraméterekkel rendelkező kis modult - mely akár lehet egy alaplapra forrasztott áramkör is - sorozatgyártásban nagyon alacsony költséggel elő lehetne állítani. Ennek megtervezése az én dolgo-

[1] A BeagleBoard egy kis teljesítményű, nyílt forráskódú, hitelkártya méretű miniszámítógép. http://beagleboard.org/

zatomnak nem része. Legjobb megoldásnak valamilyen mikrokontrolleres egységet látok. Megvalósításához egy 8 bites mikrokontrollerre, Ethernet portra, relékre, illesztőkártyára és további egységekre a feszültség és áram ellenőrzéshez lenne szükség. Ezek költsége körülbelül 5000 Ft-ot tesz ki. Kiemelném ugyanakkor, hogy ez az összeg egy kiskereskedelmi forgalomban beszerzendő egységek esetén igaz. Sorozatgyártás esetén ez a költség a felére, vagy akár a harmadára lecsökkenhet. Ugyanakkor az alábbi, közelítő költségvetési tervvel kívánom alátámasztani az intelligens jelzőfej létjogosultságát (lásd 8.2 táblázat). Százalékban kifejezve

Szükséges eszközök és kellékek megnevezése	**Hagyományos jelzőlámpás irányítás költsége [Ft]**	**Intelligens jelzőfejjel megvalósított irányítás költsége [Ft]**	**Pénzbeli megtérülés [Ft]**
SZRMKVM - J 4x1.5 mm kábel	47804	40052	7752
SZRMKVM - J 5x1.5 mm kábel	20405	-	20405
SZRMKVM - J 19x1.5 mm kábel	31680	-	31680
UTP kábel	-	18600	-18600
Logikai egység az irányításra (mikrokontroller)	-	5000	-5000
Összesen:	99889	63652	**36237**

8.2. táblázat. Költségvetési terv táblázata

ez **36,28 %**-os végső **megtakarítási költséget** jelent, az általam felhasznált és becsült adatok alapján. Mindemellett azt vegyük figyelembe, hogy ez a számítás egy átlagos T-csomópontra lett elvégezve. Tehát a csomópont méretének növekedésével a megtakarítás abszolút értéke természetesen nő, és emellett várhatóan a relatív értéke is növekedni fog. Ezen felül a költségvetés számítás nem tartalmazza az élő munkaerőben megspórolható költségeket. Egy általános forgalomirányításra épített rendszernél ugyanis a kábelek fektetése, behúzása a csatornákba sok kábel esetén sok időt, illetve emberi munkaerőt igényel. Továbbá több időt és energiát kíván az elektromos kábelek beazonosítása is, hogy még véletlenül se forduljon elő olyan eset, hogy egy adott lámpát rossz végre kötnek. A gyakorlatban ezért szokták az izzókat egyenként „kivillogtatni”, mielőtt elindítják a csomópontot.

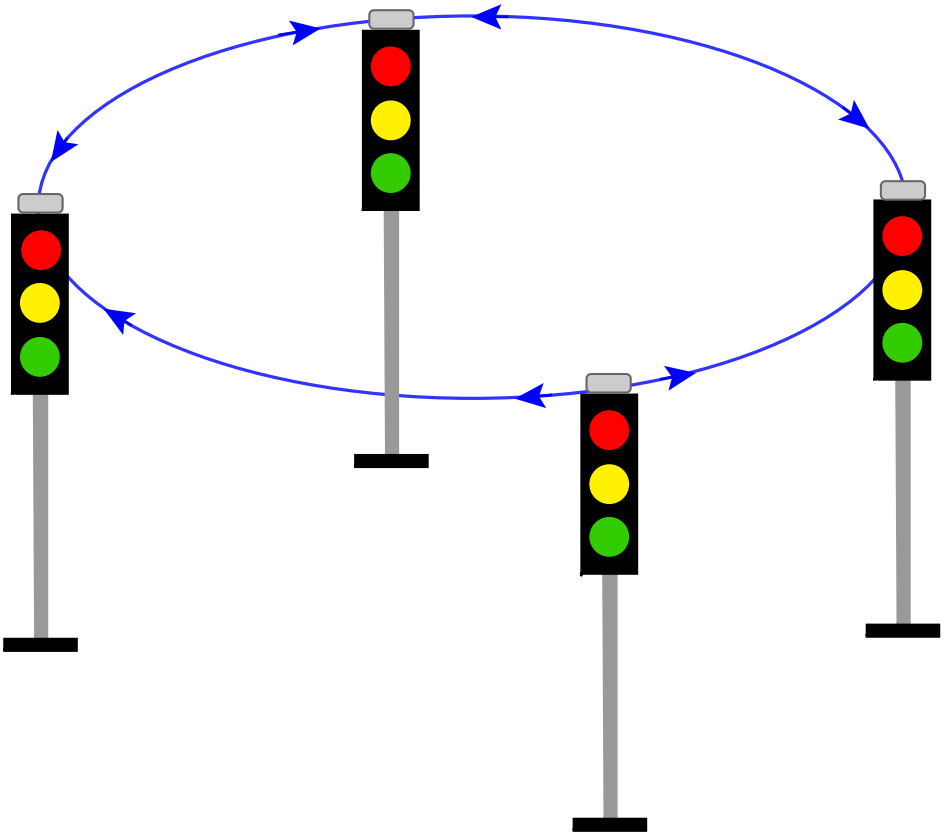
Az intelligens jelzőfejekkel vezérelt rendszernél minderre nincs szükség, mert egy elektromos kábel vezet a tápellátáshoz, és a logikai egységekbe kötött UTP kábeleket mindegy, hogy milyen sorrendbe kötjük be az irányító egységekbe. Ezek beazonosítása egyszerűen a kiosztott IP címek alapján megvalósítható.

9. fejezet

A rendszer továbbfejlesztésének lehetőségei

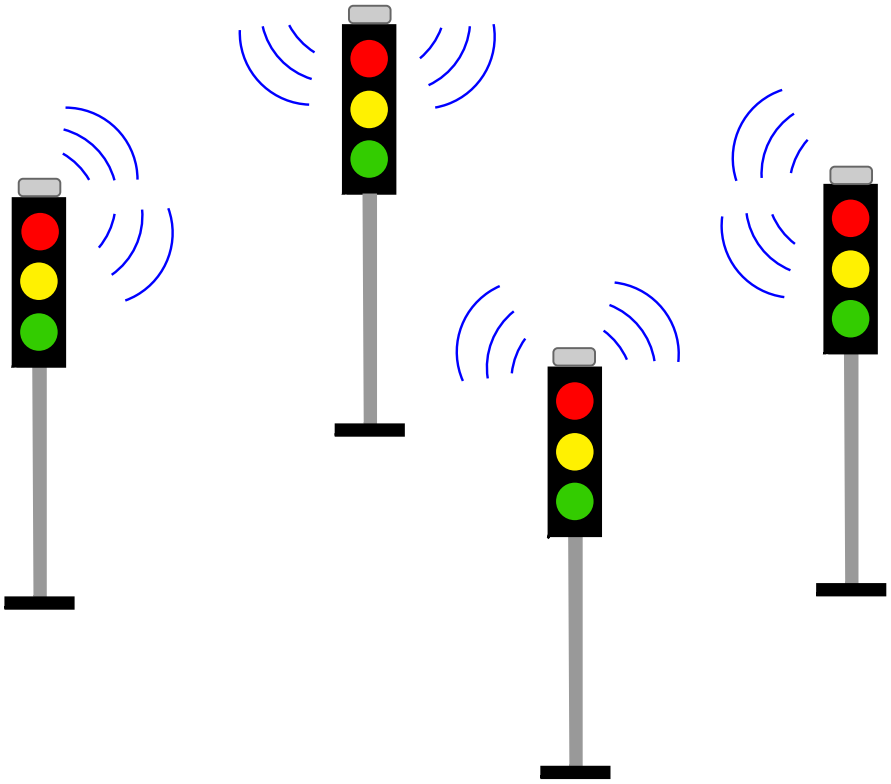
Az intelligens jelzőfejjel működtetett rendszer számos előnye van. A kábelezés nagy részét el lehet hagyni, ezzel a beruházások költségein csökkenteni lehet, továbbá a kiépítése gyorsabb és egyszerűbb, és alkalmas egymagában egyszerűbb terelések vezérlésénél a forgalom irányítására. Ellenben az én projektemet is tovább lehet még fejleszteni, ezeket foglaltam össze a következőkben.
Az általam felépített rendszerben az intelligens jelzőfej kis számítógépének alkalmazott PLC nem a konkrét feladathoz fejlesztett egység, hanem egy jóval nagyobb tudású vezérlő számítógép. A PLC helyett olyan egyszerűbb cél hardvert kellene alkalmazni, amely alkalmas a szükséges logikai műveletek elvégzésére, és megvalósítható vele az IP alapú hálózati kommunikáció. A jövőben szeretném az intelligens jelzőfejjel működő rendszert PLC-k helyett mikrokontrolleres vezérlő egységgel kiépíteni. Ehhez szükség van egy 8 bites mikrokontrollerre és a hozzá tartozó relékre, Ethernet portra, illesztőkártyára. Továbbá újra kellene tervezni a forgalomirányító berendezés és az intelligens jelzőfejek közötti kommunikációt.
A rendszer továbbfejlesztésének másik része az alkalmazott hálózati kommunikáció revíziója, amit a mikrokontrollerre való váltás magával von. A kialakított rendszerben TCP/IP protokollt alkalmaztam, amely sebessége (lévén nem real-time protokoll) nagyobb csomópontok esetén már okozhat problémát. A kommunikációs sebesség felgyorsítása érdekében az UDP (User Data Protocol) protokoll alkalmazása tűnik kézenfekvőnek a megfelelő ellenőrzés megvalósítása mellett, de akár a soros kommunikáció is a megoldások közé tartozhat az egyszerűsége miatt. Mindezek mellett érdemes lenne az intelligens jelzőfejjel működő rendszerrel elérhető megtakarításokat, különböző nagyságú csomópontok esetén is elvégezni. A gazdasági számítások pontosságát érdemes mélyebbre menő kutatásokkal, és felmérésekkel alátámasztani.
Távolabbi kilátások közé tartozik a jelzőfejek intelligenciájának olyan szintre va-

ló növelése, hogy a forgalomirányító berendezés elhagyható legyen a rendszerből. Ezzel további költségmegtakarítást lehetne elérni. A rendszerben minden jelzőfej tudná a jelzésterveket, és egymással folyamatos kommunikációban lennének. Az ellenőrzéseket is ők maguk végeznék el.



9.1. ábra. A forgalomirányító berendezés nélküli, megnövelt intelligenciájú jelzőfejek

Legvégső állapot lehetne, amikor a jelzőfejek közötti kábel kapcsolat teljes mértékben fel van számolva (lásd 9.2 ábra). Helyette rádiós kapcsolaton keresztül történne a kommunikáció. Ez a wireless kapcsolat valamilyen kis hatótávolságú rádiófrekvenciás kapcsolat kellene legyen. Legjobb megoldásnak a 2,4 Ghz-en való kommunikációt tartom, mert itt nem fordulhat olyan elő, hogy más eszköz ugyan azt a csatornát használja mint a jelzőfejek. Problémája, hogy a kapcsolat megbízhatósága a kábeles kapcsolathoz képest jelentősen gyengébb. Ellenben vezeték nélkül működő rendszerrel akár még több költséget lehetne megtakarítani a kábeleke teljes elhanyagolása miatt.



9.2. ábra. Megnövelt intelligenciájú jelzőfejek rádiós kapcsolattal

10. fejezet

Összefoglaló

Az alapötletet, miszerint létezhet-e olyan jelzőfej, mely valamilyen intelligenciát kapva képes az izzókat maga vezérelni, szakdolgozatomban bizonyítva lett. A technika rohamléptekben való fejlődésével lépést kell tartani, és az úgynevezett „okos" eszközpark skáláját ki kell terjeszteni a közlekedés irányításra is. Az intelligens jelzőfejekkel működő rendszert sikerült megvalósítanom, így azt mondhatom, hogy létjogosultságot nyert a közúti forgalomirányításban.
A rendszerben a kábelek hosszának és számának lecsökkentésével jelentős költségmegtakarítást lehet elérni, amit dolgozatomban gazdasági számításokkal igazoltam egy kisebb méretű csomópontra. A csomópont méretének növekedésével az intelligens jelzőfejek használata esetén a költségmegtakarítás növekszik. Mindemellett az alkalmazhatósága is jobb, mint a hagyományos rendszernek, mert telepítése egyszerűbb és gyorsabb.
A forgalomirányító berendezés (Actros) és az intelligens jelzőfejek között a kapcsolatot felépítettem, így képes a két eszköz egymásnak feladatokat adni és kommunikálni. Az irányítást az Actros oldaláról újra kellett építeni, illetve a PLC-re programot kellett tervezni. Ebbe bele kellett kalkulálni a forgalomirányító berendezéssel való kapcsolat ellenőrzésére szolgáló biztonsági funkciót. Továbbá az izzók ellenőrzése mind feszültség, mind áram ellenőrzésével megvalósult.
Munkám igazolja, hogy az intelligens jelzőfejjel működő forgalomirányítás mindenképpen egy hasznos tanulmány és továbbfejlesztésével érdemes foglalkozni.

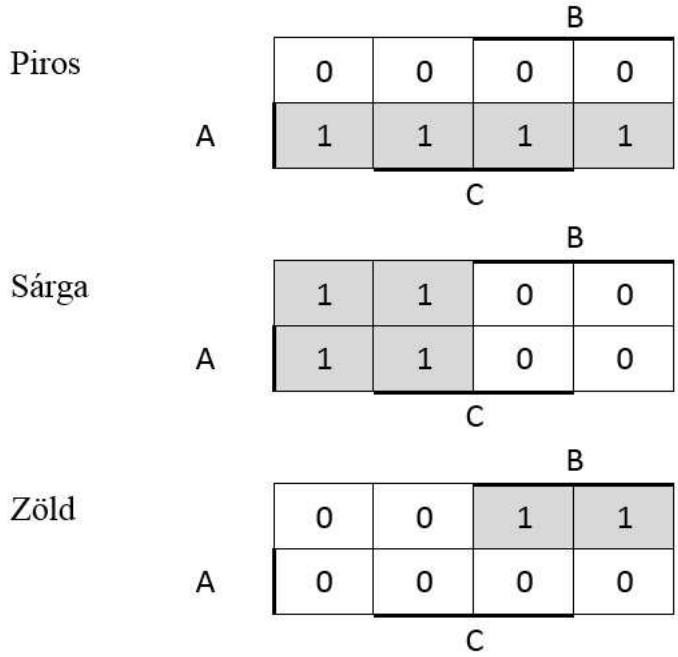
A. függelék

Mintapélda a jelzőlámpa kapcsolásra Logo Soft Comfortban

A következőkben egy mintapélda keretében fogom bemutatni Logo Soft Comfortban való tervezést és megvalósítást. A feladat a jelzőlámpa váltás, illetve az izzók működtetésének feladatát dolgozza ki. A tervezést és programozást magán a PLC-n is el lehetett volna végezni, de a kis kijelzője és a szerény vezérlőgombjai miatt elég körülményes, így inkább a Siemens által erre a célra fejlesztett Logo Soft Comfort szoftvert vettem igénybe.
Négy állapotot kellett megjelenítenem, ezek sorban: Piros, Piros-Sárga, Zöld, Sárga. Az állapotokat egy táblázatba vettem fel (lásd A.1 táblázat). A négy darab állapotot három darab izzó hajtotta végre. A feladat az volt, hogy a megfelelő időben, és a várt izzó kapcsoljon be. A problémát legegyszerűbben Karnaugh - tábla felrajzolásával tudtam megoldani (lásd A.1 táblázat).

	A	B	C	Piros	Sárga	Zöld
0.	0	0	0	0	1	0
1.	0	0	1	0	1	0
2.	0	1	0	0	0	1
3.	0	1	1	0	0	1
4.	1	0	0	1	1	0
5.	1	0	1	1	1	0
6.	1	1	0	1	0	0
7.	1	1	1	1	0	0

A.1. táblázat. Az állapotokat mutató táblázat



A.1. ábra. Karnaugh táblák

Az eredmények:

- FP = A
- FS = $\overline{B}$
- FZ = $\overline{A}$B

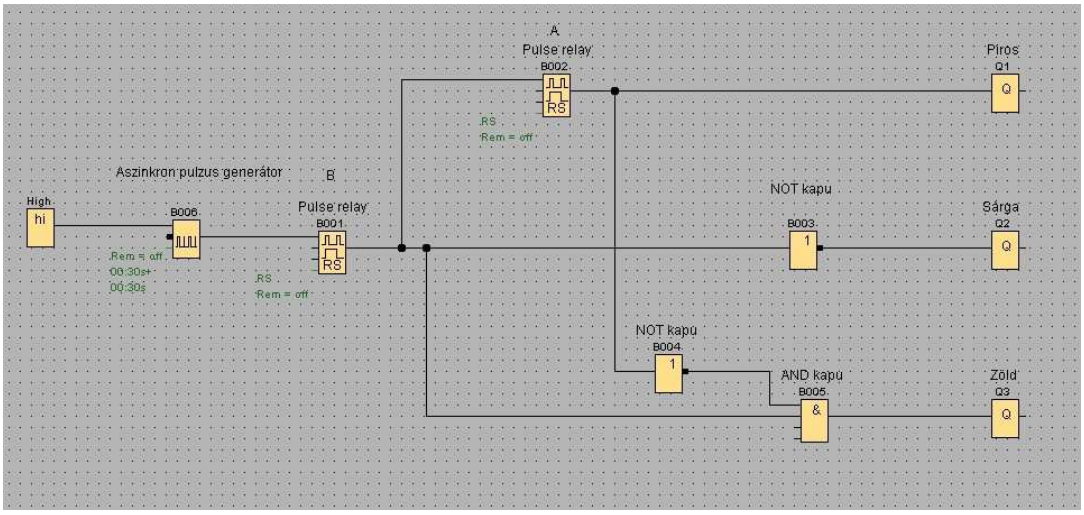
[Tarnai et al., 2011]

Az eredmények ismeretében könnyen meg tudtam rajzolni ezeket az állapotokat a Logo Soft Comfortban. A cél azonban az volt, hogy a jelzők állapotai időben változva automatikusan váltogassanak, és ne manuálisan kelljen az állapotátmeneteket végrehajtani. Ennek megoldására bevezettem a rendszerbe egy aszinkron pulzus generátort. Ez a kimenetén ugráltatja a jelet '0' és '1' között, a tulajdonságaiban megadott időintervallum szerint. Majd felhelyeztem a rajzpalettára, egy High blokkot, amely folyamatosan '1'-et adott a kimenetére, így meghajtotta a mögé kötött aszinkron pulzus generátort (lásd A.2).
A generátoron kettőt kattintva megjelenik egy ablak, ami a blokk tulajdonságait tartalmazza. Itt beállítottam a Pulse Width -re 30 Seconds-ot [s: 1/100 s], és az Interpulse Width -re is 30 Seconds-ot [s: 1/100 s]. Ezzel azt az eredményt kaptam, hogy a generátor 30 századmásodpercenként kapcsolgatta a belőle kijövő jelet, ami nem más, mint az A.1 'C' oszlopa. A 'B', és az 'A' oszlopot is elő kellett állítanom, ezért szükségem volt két 'Pulse relay' (Impulzus relé)-re. Az impulzus relé a következőképp működik. Ha '1'-et adok a bemenetére, akkor a kimenetén megjelenik az '1'. Ha ráadom a '0'-át, akkor húzva marad, és tartja az '1'-et a kimenetén, majd ha újra '1'-et adok rá, akkor megtörténik a reset funkció és a kimenete '0'-ra vált. Lényegében úgy működik, mint egy SR tároló. Egy ilyen Pulse relay-t be-raktam az aszinkron pulzus generátor mögé, így a '0','1','0','1','0','1','0','1' jelsorozatot át tudtam alakítani '0','0','1','1','0','0','1','1' jelsorozattá, amivel megkaptam az A.1 'B' oszlopát. Majd e mögé egy újabb Pulse relay-t helyeztem és így eljutottam a '0','0','0','0','1','1','1','1' jelsorozatig, ami az A.1 táblázat 'A' oszlopának felel meg (lásd A.2 ábra). Ezek után már csak a Karnaugh – táblákból (lásd A.1 táblázat) megkapott eredményeket kellett feldolgoznom.

- A piros égést megkaptam, amikor egy Output (Kimenet) blokkot felhelyeztem a rajztáblára és bekötöttem az A.1 táblázat után 'A'-nak elnevezett utolsó Pulse relay -be.
- A sárga égést megkaptam, mikor elhelyeztem egy Kimenet blokkot, és ezt egy NOT (negáció) kapuval sorba kötve az A.1 táblázat után 'B'-nek elnevezett első Pulse relay kimenetére kötöttem.

- Végül a zöld égést úgy állítottam elő, hogy egy újabb Kimenet blokkot helyeztem a palettára. Ennek bemenetét egy AND (és) kapuval kötöttem össze. Az AND kapu egyik bemenetére rákötöttem, az 'A' kimenetét egy NOT-al közbeiktatva, a másik bemenetére pedig a 'B' kimenetét kötöttem.

Az egyes blokkokat érdemes elnevezni a rendszer jobb átláthatóságának érdekében, melyet a tulajdonságok alatt tudunk megtenni. A blokkra duplán kattintva előugrik egy ablak, melynek mindig a legutolsó füle a Comment fül. Ez alatt tetszés szerinti nevet lehet változtatni, ami a blokk fölött fog megjelenni. Az elkészült programot az A.2 ábra szemlélteti.



A.2. ábra. Blokkdiagram a jelzőlámpás váltásról Logo Soft Comfortban

Az elkészült áramkör a Logo Soft Comfortban is futtatható, egy szimuláció keretében. Ezt az F3 gomb megnyomásával lehet elindítani. Ha tervezés jó, akkor a kimeneteken a lámpák a megfelelő sorrendben villannak fel. Természetesen a program LOGO!-ra is feltölthető, és ha a fizikai kapcsolat fenn áll a jelzőfejekkel, akkor az izzók megfelelő időben és sorrendben villannak majd fel.

B. függelék

A LogoJelzofej.java osztály kódsora

```
package csojan;

import java.io.IOException;
import java.net.UnknownHostException;
import java.lang.InterruptedException;
import vt.StgEbene;
import hu.edu.bme.kka.Logo;

public class LogoJelzofej{
	private Logo logo_i;
	public LogoJelzofej(String address) {
		try {
			logo_i = new Logo (address);
		} catch (UnknownHostException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
	private static final int PIROSSARGA_IDO = 2;
	private static final int SARGA_IDO = 3;
   private static final byte PIROS = 1;
   private static final byte SARGA = 2;
   private static final byte ZOLD = 4;
   private static final byte PIROSSARGA = PIROS|SARGA;
```

```
private static final byte SARGAVILLOGO = 8;
private volatile boolean zoldeges;

int kozb01 = 10;
    int kozb02 = 10;
    int kozb10 = 10;
    int kozb12 = 10;
    int kozb20 = 10;
    int kozb21 = 10;

private int bef;
public int getBefejezes() {
            return bef;
    }

    public int getKezdes() {
            return kezd;
    }
    private int kezd;

    public void belepo(){
            logo_i.setByte(0,1);
    }

    public void BEKI(int kezdes,int befejezes,final int logoszama) {
            int sec = Var.tk1.getProgSek();
            int ciklusido = Var.tk1.getZyklDauer();
            bef = befejezes;
            kezd = kezdes;

            class Szal2 implements Runnable{

            private String nevem;

            public Szal2(String _nevem){
                    nevem = _nevem;
            }

            public void run(){

                        try {
                                Thread.sleep(5);
```

```
                        if (Var.tk1.getProgZeit() % 2000 ==
                            0){            //vezerjel adas
                                logo_i.setDWord(1,10);}

                        if (Var.tk1.getProgZeit() % 1000 ==
                            0){
                        System.out.println(logoszama + "
                            PLC " + "zoldeges " +
                            logo_i.getInput(2)+" " +
                            Var.tk1.getProgZeit());} //fesz
                            ell.
                        System.out.println(" ");
                    }
                    catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        System.out.println("Szal hiba");
                        System.out.println(e.getMessage());
                    }
                    //System.out.println("Stop 2
                        "+Var.tk1.getProgZeit());
        }
}

Szal2 s1 = new Szal2("Szal 02");    //altalunk keszitett osztaly
    ami impelementalja a runnable feluletet
Thread sz1 = new Thread(s1);        //3 futo szal letrehozasa,
    es atadjuk neki az elobb keszitett osztalyt
//System.out.println("Start 2 "+Var.tk1.getProgZeit());
sz1.start();                //3 futo szal letrehozasa, es
    atadjuk neki az elobb keszitett osztalyt


        //Kozbenso ido ellenorzes

        kozb01 =
            Math.abs((Var.logo.getKezdes()-Var.logo1.getBefejezes()));
        kozb02 =
            Math.abs((Var.logo.getKezdes()-Var.logo2.getBefejezes()));
        kozb10 =
            Math.abs((Var.logo1.getKezdes()-Var.logo.getBefejezes()));
        kozb12 =
            Math.abs((Var.logo1.getKezdes()-Var.logo2.getBefejezes()));
```

```
            kozb20 =
                Math.abs((Var.logo2.getKezdes()-Var.logo.getBefejezes()));
            kozb21 =
                Math.abs((Var.logo2.getKezdes()-Var.logo1.getBefejezes()));

            if(kozb01>3 & kozb02>3 & kozb10>3 & kozb12>3 & kozb20>3 &
                kozb21>3){
                    if (Var.tk1.getProgZeit()% 10000 == 0){
                            System.out.println("A kozbenso idok
                                rendben vannak");
                            System.out.println(" ");
                    }



                    if(sec>=kezdes && sec< kezdes+PIROSSARGA_IDO){
                            set(PIROSSARGA, logoszama);
                            }
                    if(sec == kezdes+PIROSSARGA_IDO){
                            set(ZOLD, logoszama);
                            }
                    if(sec>= befejezes && sec < befejezes+SARGA_IDO){
                            set(SARGA, logoszama);
                            }
                    if(sec == befejezes+SARGA_IDO){
                            set(PIROS, logoszama);
                            }

                    }

            else{
                    if (Var.tk1.getProgZeit()% 3000 == 0){
                            System.out.println("A kozbenso ido valahol
                                NEM OK - ezert sarga villogo!");
                    }
                    logo_i.closeConnection();
                    }
            }


    private void set(final int szin, final int logosz){ //Runnable
        interface implementalasa
```

```
        class Szal implements Runnable{
                        private int szin_ = szin;
                        private String nevem;

                        public Szal(String _nevem){
                                nevem = _nevem;
                        }

                        public void run(){

                                        try {
                                                Thread.sleep(5);
                                                logo_i.setByte(0,szin_);

                                        } catch (InterruptedException e) {
                                                // TODO Auto-generated catch
                                                    block
                                                e.printStackTrace();
                                                System.out.println("Szal
                                                    hiba");
                                                System.out.println(e.getMessage());
                                        }
                                //System.out.println("Stop 1
                                    "+Var.tk1.getProgZeit());
                                }
                        }

                        Szal s0 = new Szal("Szal 00"); //altalunk keszitett
                            osztaly ami impelementalja a runnable feluletet
                        Thread sz0 = new Thread(s0);        //3 futo szal
                            letrehozasa, es atadjuk neki az elobb keszitett
                            osztalyt
                        //System.out.println("Start 1 "+Var.tk1.getProgZeit());
                        sz0.start();                //3 futo szal letrehozasa,
                            es atadjuk neki az elobb keszitett osztalyt

        }

}
```

Irodalomjegyzék

L. Bodó. A jelzőlámpás irányítás villamos munkáinak terve. *Swarco*, Texreport, 2008. unpublished. v, 55, 56, 57

G. Debreczeni. Közúti Forgalomtechnika. *BME KUKG*, Texreport, 2013. URL kukg.bme.hu/oktatas/bsc/segedletek/BMEKOKUA209/1_segedlet.pdf. iv, v, 2, 3, 4, 46, 47

J. Juhász. Magyar Értelmező Kéziszótár. *Akadémia Kiadó*, Book, 1972. 40

Á. Ludvig. Siemens Logo 0BA7 kommunikációs beállítása a libnodave szoftverkönyvtár alkalmazásához. *BME KJIT*, Texreport, 2013. unpublished. iv, 22, 23, 24, 26, 27, 44, 45

T. Luspay, T. Tettamanti, and I. Varga. Forgalomirányítás, Közúti járműforgalom modellezése és irányítása. *Typotex Kiadó, Budapest*, Book, 2011. ISBN 978-963-279-665-9. 2

H. Sakai and T. Kawamura. Light-Emitting Diode. *Patent number: 4698730*, Patent, 1987. URL uspto.gov/inventors/patents.jsp. 4

Siemens. Logo Soft Comfort. *User Documentation*, Texreport, 1999. 13, 14, 21

Siemens. LOGO V5 manual. *Manual*, Booklet, 2009. iv, 6, 10, 11, 12, 21

G. Tarnai, J. Bokor, B. Sághi, E. Baranyi, and T. Bécsi. Irányítástechnika I. *Typotex*, Book, 2011. 67

T. Tettamanti. ACTROS VTC 3000. *BME KJIT*, Texreport, 2010. URL kjit.bme.hu/images/stories/targyak/kozutir2/actros_vtc_3000.pdf. 15

T. Tettamanti and J. Polgár. Forgalomtechnikai kód az ACTROS VTC 3000 forgalomirányító berendezésben . *BME KJIT*, Texreport, 2010. URL kjit.bme.hu/images/stories/targyak/kozutir2/java_vt_actros.pdf. iv, 17, 31